NAME

ld − The GNU linker

SYNOPSIS

ld [**options**] *objfile* ...

DESCRIPTION

ld combines a number of object and archive files, relocates their data and ties up symbol references. Usually the last step in compiling a program is to run **ld**.

ld accepts Linker Command Language files written in a superset of AT&T's Link Editor Command Language syntax, to provide explicit and total control over the linking process.

This man page does not describe the command language; see the **ld** entry in `info` for full details on the command language and on other aspects of the GNU linker.

This version of **ld** uses the general purpose BFD libraries to operate on object files. This allows **ld** to read, combine, and write object files in many different formats−−−for example, COFF or `a.out`. Different formats may be linked together to produce any available kind of object file.

Aside from its flexibility, the GNU linker is more helpful than other linkers in providing diagnostic information. Many linkers abandon execution immediately upon encountering an error; whenever possible, **ld** continues executing, allowing you to identify other errors (or, in some cases, to get an output file in spite of the error).

The GNU linker **ld** is meant to cover a broad range of situations, and to be as compatible as possible with other linkers. As a result, you have many choices to control its behavior.

OPTIONS

The linker supports a plethora of command-line options, but in actual practice few of them are used in any particular context. For instance, a frequent use of **ld** is to link standard Unix object files on a standard, supported Unix system. On such a system, to link a file `hello.o`:

```
ld −o <output> /lib/crt0.o hello.o −lc
```

This tells **ld** to produce a file called *output* as the result of linking the file `/lib/crt0.o` with `hello.o` and the library `libc.a`, which will come from the standard search directories. (See the discussion of the **−l** option below.)

Some of the command-line options to **ld** may be specified at any point in the command line. However, options which refer to files, such as **−l** or **−T**, cause the file to be read at the point at which the option appears in the command line, relative to the object files and other file options. Repeating non-file options with a different argument will either have no further effect, or override prior occurrences (those further to the left on the command line) of that option. Options which may be meaningfully specified more than once are noted in the descriptions below.

Non-option arguments are object files or archives which are to be linked together. They may follow, precede, or be mixed in with command-line options, except that an object file argument may not be placed between an option and its argument.

Usually the linker is invoked with at least one object file, but you can specify other forms of binary input files using **−l**, **−R**, and the script command language. If *no* binary input files at all are specified, the linker does not produce any output, and issues the message **No input files**.

If the linker cannot recognize the format of an object file, it will assume that it is a linker script. A script specified in this way augments the main linker script used for the link (either the default linker script or the one specified by using **−T**). This feature permits the linker to link against a file which appears to be an object or an archive, but actually merely defines some symbol values, or uses `INPUT` or `GROUP` to load other objects. Specifying a script in this way merely augments the main linker script, with the extra commands placed after the main script; use the **−T** option to replace the default linker script entirely, but note the effect of the `INSERT` command.

For options whose names are a single letter, option arguments must either follow the option letter without

intervening whitespace, or be given as separate arguments immediately following the option that requires them.

For options whose names are multiple letters, either one dash or two can precede the option name; for example, **−trace−symbol** and **−−trace−symbol** are equivalent. Note−−−there is one exception to this rule. Multiple letter options that start with a lower case 'o' can only be preceded by two dashes. This is to reduce confusion with the **−o** option. So for example **−omagic** sets the output file name to **magic** whereas **−−omagic** sets the NMAGIC flag on the output.

Arguments to multiple-letter options must either be separated from the option name by an equals sign, or be given as separate arguments immediately following the option that requires them. For example, **−−trace−symbol foo** and **−−trace−symbol=foo** are equivalent. Unique abbreviations of the names of multiple-letter options are accepted.

Note−−−if the linker is being invoked indirectly, via a compiler driver (e.g. **gcc**) then all the linker command-line options should be prefixed by **−Wl,** (or whatever is appropriate for the particular compiler driver) like this:

```
gcc -Wl,--start-group foo.o bar.o -Wl,--end-group
```

This is important, because otherwise the compiler driver program may silently drop the linker options, resulting in a bad link. Confusion may also arise when passing options that require values through a driver, as the use of a space between option and argument acts as a separator, and causes the driver to pass only the option to the linker and the argument to the compiler. In this case, it is simplest to use the joined forms of both single− and multiple-letter options, such as:

```
gcc foo.o bar.o -Wl,-eENTRY -Wl,-Map=a.map
```

Here is a table of the generic command-line switches accepted by the GNU linker:

@file
: Read command-line options from *file*. The options read are inserted in place of the original *@file* option. If *file* does not exist, or cannot be read, then the option will be treated literally, and not removed.

  Options in *file* are separated by whitespace. A whitespace character may be included in an option by surrounding the entire option in either single or double quotes. Any character (including a backslash) may be included by prefixing the character to be included with a backslash. The *file* may itself contain additional *@file* options; any such options will be processed recursively.

−a *keyword*
: This option is supported for HP/UX compatibility. The *keyword* argument must be one of the strings **archive**, **shared**, or **default**. **−aarchive** is functionally equivalent to **−Bstatic**, and the other two keywords are functionally equivalent to **−Bdynamic**. This option may be used any number of times.

−−audit *AUDITLIB*
: Adds *AUDITLIB* to the `DT_AUDIT` entry of the dynamic section. *AUDITLIB* is not checked for existence, nor will it use the DT_SONAME specified in the library. If specified multiple times `DT_AUDIT` will contain a colon separated list of audit interfaces to use. If the linker finds an object with an audit entry while searching for shared libraries, it will add a corresponding `DT_DEPAUDIT` entry in the output file. This option is only meaningful on ELF platforms supporting the rtld-audit interface.

−b *input-format*
−−format=*input-format*
: **ld** may be configured to support more than one kind of object file. If your **ld** is configured this way, you can use the **−b** option to specify the binary format for input object files that follow this option on the command line. Even when **ld** is configured to support alternative object formats, you don't usually need to specify this, as **ld** should be configured to expect as a default input format the most usual format on each machine. *input-format* is a text string, the name of a particular format supported by the BFD libraries. (You can list the available binary formats with **objdump −i**.)

You may want to use this option if you are linking files with an unusual binary format. You can also use **−b** to switch formats explicitly (when linking object files of different formats), by including **−b** *input-format* before each group of object files in a particular format.

The default format is taken from the environment variable `GNUTARGET`.

You can also define the input format from a script, using the command `TARGET`;

−c *MRI-commandfile*
−−mri−script=*MRI-commandfile*
For compatibility with linkers produced by MRI, **ld** accepts script files written in an alternate, restricted command language, described in the MRI Compatible Script Files section of GNU ld documentation. Introduce MRI script files with the option **−c**; use the **−T** option to run linker scripts written in the general-purpose **ld** scripting language. If *MRI-cmdfile* does not exist, **ld** looks for it in the directories specified by any **−L** options.

−d
−dc
−dp
These three options are equivalent; multiple forms are supported for compatibility with other linkers. They assign space to common symbols even if a relocatable output file is specified (with **−r**). The script command `FORCE_COMMON_ALLOCATION` has the same effect.

−−depaudit *AUDITLIB*
−P *AUDITLIB*
Adds *AUDITLIB* to the `DT_DEPAUDIT` entry of the dynamic section. *AUDITLIB* is not checked for existence, nor will it use the DT_SONAME specified in the library. If specified multiple times `DT_DEPAUDIT` will contain a colon separated list of audit interfaces to use. This option is only meaningful on ELF platforms supporting the rtld-audit interface. The −P option is provided for Solaris compatibility.

−−enable−linker−version
Enables the `LINKER_VERSION` linker script directive, described in **Output Section Data**. If this directive is used in a linker script and this option has been enabled then a string containing the linker version will be inserted at the current point.

Note − this location of this option on the linker command line is significant. It will only affect linker scripts that come after it on the command line, or which are built into the linker.

−−disable−linker−version
Disables the `LINKER_VERSION` linker script directive, so that it does not insert a version string. This is the default.

−−enable−non−contiguous−regions
This option avoids generating an error if an input section does not fit a matching output section. The linker tries to allocate the input section to subseque nt matching output sections, and generates an error only if no output section is large enough. This is useful when several non-contiguous memory regions are available and the input section does not require a particular one. The order in which input sections are evaluated does not change, for instance:

```
MEMORY {
  MEM1 (rwx) : ORIGIN = 0x1000, LENGTH = 0x14
  MEM2 (rwx) : ORIGIN = 0x1000, LENGTH = 0x40
  MEM3 (rwx) : ORIGIN = 0x2000, LENGTH = 0x40
}
SECTIONS {
  mem1 : { *(.data.*); } > MEM1
  mem2 : { *(.data.*); } > MEM2
  mem3 : { *(.data.*); } > MEM3
}

with input sections:
.data.1: size 8
.data.2: size 0x10
.data.3: size 4

results in .data.1 affected to mem1, and .data.2 and .data.3
affected to mem2, even though .data.3 would fit in mem3.
```

This option is incompatible with INSERT statements because it changes the way input sections are mapped to output sections.

--enable-non-contiguous-regions-warnings
This option enables warnings when `--enable-non-contiguous-regions` allows possibly unexpected matches in sections mapping, potentially leading to silently discarding a section instead of failing because it does not fit any output region.

-e *entry*
--entry=*entry*
Use *entry* as the explicit symbol for beginning execution of your program, rather than the default entry point. If there is no symbol named *entry*, the linker will try to parse *entry* as a number, and use that as the entry address (the number will be interpreted in base 10; you may use a leading **0x** for base 16, or a leading **0** for base 8).

--exclude-libs *lib*,*lib*,...
Specifies a list of archive libraries from which symbols should not be automatically exported. The library names may be delimited by commas or colons. Specifying `--exclude-libs ALL` excludes symbols in all archive libraries from automatic export. This option is available only for the i386 PE targeted port of the linker and for ELF targeted ports. For i386 PE, symbols explicitly listed in a .def file are still exported, regardless of this option. For ELF targeted ports, symbols affected by this option will be treated as hidden.

--exclude-modules-for-implib *module*,*module*,...
Specifies a list of object files or archive members, from which symbols should not be automatically exported, but which should be copied wholesale into the import library being generated during the link. The module names may be delimited by commas or colons, and must match exactly the filenames used by **ld** to open the files; for archive members, this is simply the member name, but for object files the name listed must include and match precisely any path used to specify the input file on the linker's command-line. This option is available only for the i386 PE targeted port of the linker. Symbols explicitly listed in a .def file are still exported, regardless of this option.

-E
--export-dynamic
--no-export-dynamic
When creating a dynamically linked executable, using the **-E** option or the **--export-dynamic** option causes the linker to add all symbols to the dynamic symbol table. The dynamic symbol table is the set of symbols which are visible from dynamic objects at run time.

If you do not use either of these options (or use the **--no-export-dynamic** option to restore the default behavior), the dynamic symbol table will normally contain only those symbols which are referenced by some dynamic object mentioned in the link.

If you use `dlopen` to load a dynamic object which needs to refer back to the symbols defined by the program, rather than some other dynamic object, then you will probably need to use this option when linking the program itself.

You can also use the dynamic list to control what symbols should be added to the dynamic symbol table if the output format supports it. See the description of **--dynamic-list**.

Note that this option is specific to ELF targeted ports. PE targets support a similar function to export all symbols from a DLL or EXE; see the description of **--export-all-symbols** below.

--export-dynamic-symbol=*glob*
When creating a dynamically linked executable, symbols matching *glob* will be added to the dynamic symbol table. When creating a shared library, references to symbols matching *glob* will not be bound to the definitions within the shared library. This option is a no-op when creating a shared library and **-Bsymbolic** or **--dynamic-list** are not specified. This option is only meaningful on ELF platforms which support shared libraries.

--export-dynamic-symbol-list=*file*
Specify a **--export-dynamic-symbol** for each pattern in the file. The format of the file is the same as the version node without scope and node name. See **VERSION** for more information.

-EB
Link big-endian objects. This affects the default output format.

-EL
Link little-endian objects. This affects the default output format.

-f *name*
--auxiliary=*name*
When creating an ELF shared object, set the internal DT_AUXILIARY field to the specified name. This tells the dynamic linker that the symbol table of the shared object should be used as an auxiliary filter on the symbol table of the shared object *name*.

If you later link a program against this filter object, then, when you run the program, the dynamic linker will see the DT_AUXILIARY field. If the dynamic linker resolves any symbols from the filter object, it will first check whether there is a definition in the shared object *name*. If there is one, it will be used instead of the definition in the filter object. The shared object *name* need not exist. Thus the shared object *name* may be used to provide an alternative implementation of certain functions, perhaps for debugging or for machine-specific performance.

This option may be specified more than once. The DT_AUXILIARY entries will be created in the order in which they appear on the command line.

-F *name*
--filter=*name*
When creating an ELF shared object, set the internal DT_FILTER field to the specified name. This tells the dynamic linker that the symbol table of the shared object which is being created should be used as a filter on the symbol table of the shared object *name*.

If you later link a program against this filter object, then, when you run the program, the dynamic linker will see the DT_FILTER field. The dynamic linker will resolve symbols according to the symbol table of the filter object as usual, but it will actually link to the definitions found in the shared object *name*. Thus the filter object can be used to select a subset of the symbols provided by the object *name*.

Some older linkers used the **-F** option throughout a compilation toolchain for specifying object-file format for both input and output object files. The GNU linker uses other mechanisms for this purpose: the **-b**, **--format**, **--oformat** options, the `TARGET` command in linker scripts, and the `GNUTARGET`

environment variable. The GNU linker will ignore the **−F** option when not creating an ELF shared object.

−fini=*name*
When creating an ELF executable or shared object, call NAME when the executable or shared object is unloaded, by setting DT_FINI to the address of the function. By default, the linker uses `_fini` as the function to call.

−g Ignored. Provided for compatibility with other tools.

−G *value*
−−gpsize=*value*
Set the maximum size of objects to be optimized using the GP register to *size*. This is only meaningful for object file formats such as MIPS ELF that support putting large and small objects into different sections. This is ignored for other object file formats.

−h *name*
−soname=*name*
When creating an ELF shared object, set the internal DT_SONAME field to the specified name. When an executable is linked with a shared object which has a DT_SONAME field, then when the executable is run the dynamic linker will attempt to load the shared object specified by the DT_SONAME field rather than using the file name given to the linker.

−i Perform an incremental link (same as option **−r**).

−init=*name*
When creating an ELF executable or shared object, call NAME when the executable or shared object is loaded, by setting DT_INIT to the address of the function. By default, the linker uses `_init` as the function to call.

−l *namespec*
−−library=*namespec*
Add the archive or object file specified by *namespec* to the list of files to link. This option may be used any number of times. If *namespec* is of the form *:filename*, **ld** will search the library path for a file called *filename*, otherwise it will search the library path for a file called *libnamespec.a*.

On systems which support shared libraries, **ld** may also search for files other than *libnamespec.a*. Specifically, on ELF and SunOS systems, **ld** will search a directory for a library called *libnamespec.so* before searching for one called *libnamespec.a*. (By convention, a `.so` extension indicates a shared library.) Note that this behavior does not apply to *:filename*, which always specifies a file called *filename*.

The linker will search an archive only once, at the location where it is specified on the command line. If the archive defines a symbol which was undefined in some object which appeared before the archive on the command line, the linker will include the appropriate file(s) from the archive. However, an undefined symbol in an object appearing later on the command line will not cause the linker to search the archive again.

See the **−(** option for a way to force the linker to search archives multiple times.

You may list the same archive multiple times on the command line.

This type of archive searching is standard for Unix linkers. However, if you are using **ld** on AIX, note that it is different from the behaviour of the AIX linker.

−L *searchdir*
−−library−path=*searchdir*
Add path *searchdir* to the list of paths that **ld** will search for archive libraries and **ld** control scripts. You may use this option any number of times. The directories are searched in the order in which they are specified on the command line. Directories specified on the command line are searched before the default directories. All **−L** options apply to all **−l** options, regardless of the order in which the options appear. **−L** options do not affect how **ld** searches for a linker script unless **−T** option is specified.

If *searchdir* begins with = or `$SYSROOT`, then this prefix will be replaced by the *sysroot prefix*, controlled by the **--sysroot** option, or specified when the linker is configured.

The default set of paths searched (without being specified with **-L**) depends on which emulation mode **ld** is using, and in some cases also on how it was configured.

The paths can also be specified in a link script with the `SEARCH_DIR` command. Directories specified this way are searched at the point in which the linker script appears in the command line.

-m *emulation*
Emulate the *emulation* linker. You can list the available emulations with the **--verbose** or **-V** options.

If the **-m** option is not used, the emulation is taken from the `LDEMULATION` environment variable, if that is defined.

Otherwise, the default emulation depends upon how the linker was configured.

--remap-inputs=*pattern=filename*
--remap-inputs-file=*file*
These options allow the names of input files to be changed before the linker attempts to open them. The option **--remap-inputs=foo.o=bar.o** will cause any attempt to load a file called *foo.o* to instead try to load a file called *bar.o*. Wildcard patterns are permitted in the first filename, so **--remap-inputs=foo*.o=bar.o** will rename any input file that matches *foo*.o* to *bar.o*.

An alternative form of the option **--remap-inputs-file=filename** allows the remappings to be read from a file. Each line in the file can contain a single remapping. Blank lines are ignored. Anything from a hash character (**#**) to the end of a line is considered to be a comment and is also ignored. The mapping pattern can be separated from the filename by whitespace or an equals (**=**) character.

The options can be specified multiple times. Their contents accumulate. The remappings will be processed in the order in which they occur on the command line, and if they come from a file, in the order in which they occur in the file. If a match is made, no further checking for that filename will be performed.

If the replacement filename is */dev/null* or just *NUL* then the remapping will actually cause the input file to be ignored. This can be a convenient way to experiment with removing input files from a complicated build environment.

Note that this option is position dependent and only affects filenames that come after it on the command line. Thus:

```
ld foo.o --remap-inputs=foo.o=bar.o
```

Will have no effect, whereas:

```
ld --remap-inputs=foo.o=bar.o foo.o
```

Will rename the input file *foo.o* to *bar.o*.

Note – these options also affect files referenced by *INPUT* statements in linker scripts. But since linker scripts are processed after the entire command line is read, the position of the remap options on the command line is not significant.

If the **verbose** option is enabled then any mappings that match will be reported, although again the **verbose** option needs to be enabled on the command line *before* the remaped filenames appear.

If the **-Map** or **--print-map** options are enabled then the remapping list will be included in the map output.

-M
--print-map
Print a link map to the standard output. A link map provides information about the link, including the following:

- Where object files are mapped into memory.
- How common symbols are allocated.
- All archive members included in the link, with a mention of the symbol which caused the archive member to be brought in.
- The values assigned to symbols.

  Note – symbols whose values are computed by an expression which involves a reference to a previous value of the same symbol may not have correct result displayed in the link map. This is because the linker discards intermediate results and only retains the final value of an expression. Under such circumstances the linker will display the final value enclosed by square brackets. Thus for example a linker script containing:

```
foo = 1
foo = foo * 4
foo = foo + 8
```

  will produce the following output in the link map if the **–M** option is used:

```
0x00000001                  foo = 0x1
[0x0000000c]                  foo = (foo * 0x4)
[0x0000000c]                  foo = (foo + 0x8)
```

  See **Expressions** for more information about expressions in linker scripts.

- How GNU properties are merged.

  When the linker merges input .note.gnu.property sections into one output .note.gnu.property section, some properties are removed or updated. These actions are reported in the link map. For example:

```
Removed property 0xc0000002 to merge foo.o (0x1) and bar.o (not found)
```

  This indicates that property 0xc0000002 is removed from output when merging properties in *foo.o*, whose property 0xc0000002 value is 0x1, and *bar.o*, which doesn't have property 0xc0000002.

```
Updated property 0xc0010001 (0x1) to merge foo.o (0x1) and bar.o (0x1)
```

  This indicates that property 0xc0010001 value is updated to 0x1 in output when merging properties in *foo.o*, whose 0xc0010001 property value is 0x1, and *bar.o*, whose 0xc0010001 property value is 0x1.

- On some ELF targets, a list of fixups inserted by **––relax**

```
foo.o: Adjusting branch at 0x00000008 towards "far" in section .text
```

  This indicates that the branch at 0x00000008 in foo.o, targeting the symbol "far" in section .text, has been replaced by a trampoline.

––print–map–discarded
––no–print–map–discarded
Print (or do not print) the list of discarded and garbage collected sections in the link map. Enabled by default.

––print–map–locals
––no–print–map–locals
Print (or do not print) local symbols in the link map. Local symbols will have the text **(local)** printed before their name, and will be listed after all of the global symbols in a given section. Temporary local symbols (typically those that start with **.L**) will not be included in the output. Disabled by default.

−n
−−nmagic
Turn off page alignment of sections, and disable linking against shared libraries. If the output format supports Unix style magic numbers, mark the output as `NMAGIC`.

−N
−−omagic
Set the text and data sections to be readable and writable. Also, do not page-align the data segment, and disable linking against shared libraries. If the output format supports Unix style magic numbers, mark the output as `OMAGIC`. Note: Although a writable text section is allowed for PE-COFF targets, it does not conform to the format specification published by Microsoft.

−−no−omagic
This option negates most of the effects of the **−N** option. It sets the text section to be read-only, and forces the data segment to be page-aligned. Note − this option does not enable linking against shared libraries. Use **−Bdynamic** for this.

−o *output*
−−output=*output*
Use *output* as the name for the program produced by **ld**; if this option is not specified, the name *a.out* is used by default. The script command `OUTPUT` can also specify the output file name.

−−dependency−file=*depfile*
Write a *dependency file* to *depfile*. This file contains a rule suitable for `make` describing the output file and all the input files that were read to produce it. The output is similar to the compiler's output with **−M −MP**. Note that there is no option like the compiler's **−MM**, to exclude "system files" (which is not a well-specified concept in the linker, unlike "system headers" in the compiler). So the output from **−−dependency−file** is always specific to the exact state of the installation where it was produced, and should not be copied into distributed makefiles without careful editing.

−O *level*
If *level* is a numeric values greater than zero **ld** optimizes the output. This might take significantly longer and therefore probably should only be enabled for the final binary. At the moment this option only affects ELF shared library generation. Future releases of the linker may make more use of this option. Also currently there is no difference in the linker's behaviour for different non-zero values of this option. Again this may change with future releases.

−plugin *name*
Involve a plugin in the linking process. The *name* parameter is the absolute filename of the plugin. Usually this parameter is automatically added by the complier, when using link time optimization, but users can also add their own plugins if they so wish.

Note that the location of the compiler originated plugins is different from the place where the **ar**, **nm** and **ranlib** programs search for their plugins. In order for those commands to make use of a compiler based plugin it must first be copied into the *${libdir}/bfd−plugins* directory. All gcc based linker plugins are backward compatible, so it is sufficient to just copy in the newest one.

−−push−state
The **−−push−state** allows one to preserve the current state of the flags which govern the input file handling so that they can all be restored with one corresponding **−−pop−state** option.

The option which are covered are: **−Bdynamic**, **−Bstatic**, **−dn**, **−dy**, **−call_shared**, **−non_shared**, **−static**, **−N**, **−n**, **−−whole−archive**, **−−no−whole−archive**, **−r**, **−Ur**, **−−copy−dt−needed−entries**, **−−no−copy−dt−needed−entries**, **−−as−needed**, **−−no−as−needed**, and **−a**.

One target for this option are specifications for *pkg-config*. When used with the **−−libs** option all possibly needed libraries are listed and then possibly linked with all the time. It is better to return something as follows:

```
−Wl,−−push−state,−−as−needed −libone −libtwo −Wl,−−pop−state
```

−−pop−state
Undoes the effect of −−push−state, restores the previous values of the flags governing input file handling.

−q
−−emit−relocs
Leave relocation sections and contents in fully linked executables. Post link analysis and optimization tools may need this information in order to perform correct modifications of executables. This results in larger executables.

This option is currently only supported on ELF platforms.

−−force−dynamic
Force the output file to have dynamic sections. This option is specific to VxWorks targets.

−r
−−relocatable
Generate relocatable output−−−i.e., generate an output file that can in turn serve as input to **ld**. This is often called *partial linking*. As a side effect, in environments that support standard Unix magic numbers, this option also sets the output file's magic number to `OMAGIC`. If this option is not specified, an absolute file is produced. When linking C++ programs, this option *will not* resolve references to constructors; to do that, use **−Ur**.

When an input file does not have the same format as the output file, partial linking is only supported if that input file does not contain any relocations. Different output formats can have further restrictions; for example some `a.out`−based formats do not support partial linking with input files in other formats at all.

This option does the same thing as **−i**.

−R *filename*
−−just−symbols=*filename*
Read symbol names and their addresses from *filename*, but do not relocate it or include it in the output. This allows your output file to refer symbolically to absolute locations of memory defined in other programs. You may use this option more than once.

For compatibility with other ELF linkers, if the **−R** option is followed by a directory name, rather than a file name, it is treated as the **−rpath** option.

−s
−−strip−all
Omit all symbol information from the output file.

−S
−−strip−debug
Omit debugger symbol information (but not all symbols) from the output file.

−−strip−discarded
−−no−strip−discarded
Omit (or do not omit) global symbols defined in discarded sections. Enabled by default.

−t
−−trace
Print the names of the input files as **ld** processes them. If **−t** is given twice then members within archives are also printed. **−t** output is useful to generate a list of all the object files and scripts involved in linking, for example, when packaging files for a linker bug report.

−T *scriptfile*
−−script=*scriptfile*
Use *scriptfile* as the linker script. This script replaces **ld**'s default linker script (rather than adding to it), unless the script contains `INSERT`, so *commandfile* must specify everything necessary to describe the output file. If *scriptfile* does not exist in the current directory, `ld` looks for it in the directories

specified by any preceding **−L** options. Multiple **−T** options accumulate.

−dT *scriptfile*
−−default−script=*scriptfile*
Use *scriptfile* as the default linker script.

This option is similar to the **−−script** option except that processing of the script is delayed until after the rest of the command line has been processed. This allows options placed after the **−−default−script** option on the command line to affect the behaviour of the linker script, which can be important when the linker command line cannot be directly controlled by the user. (eg because the command line is being constructed by another tool, such as **gcc**).

−u *symbol*
−−undefined=*symbol*
Force *symbol* to be entered in the output file as an undefined symbol. Doing this may, for example, trigger linking of additional modules from standard libraries. **−u** may be repeated with different option arguments to enter additional undefined symbols. This option is equivalent to the `EXTERN` linker script command.

If this option is being used to force additional modules to be pulled into the link, and if it is an error for the symbol to remain undefined, then the option **−−require−defined** should be used instead.

−−require−defined=*symbol*
Require that *symbol* is defined in the output file. This option is the same as option **−−undefined** except that if *symbol* is not defined in the output file then the linker will issue an error and exit. The same effect can be achieved in a linker script by using `EXTERN`, `ASSERT` and `DEFINED` together. This option can be used multiple times to require additional symbols.

−Ur
For programs that do not use constructors or destructors, or for ELF based systems this option is equivalent to **−r**: it generates relocatable output−−−i.e., an output file that can in turn serve as input to **ld**. For other binaries however the **−Ur** option is similar to **−r** but it also resolves references to constructors and destructors.

For those systems where **−r** and **−Ur** behave differently, it does not work to use **−Ur** on files that were themselves linked with **−Ur**; once the constructor table has been built, it cannot be added to. Use **−Ur** only for the last partial link, and **−r** for the others.

−−orphan−handling=*MODE*
Control how orphan sections are handled. An orphan section is one not specifically mentioned in a linker script.

MODE can have any of the following values:

`place`
Orphan sections are placed into a suitable output section following the strategy described in **Orphan Sections**. The option **−−unique** also affects how sections are placed.

`discard`
All orphan sections are discarded, by placing them in the **/DISCARD/** section.

`warn`
The linker will place the orphan section as for `place` and also issue a warning.

`error`
The linker will exit with an error if any orphan section is found.

The default if **−−orphan−handling** is not given is `place`.

−−unique[=*SECTION***]**
Creates a separate output section for every input section matching *SECTION*, or if the optional wildcard *SECTION* argument is missing, for every orphan input section. An orphan section is one not specifically mentioned in a linker script. You may use this option multiple times on the command line;

It prevents the normal merging of input sections with the same name, overriding output section assignments in a linker script.

−v
−−version
−V Display the version number for **ld**. The **−V** option also lists the supported emulations. See also the description of the **−−enable−linker−version** in **Options,,Command−line Options** which can be used to insert the linker version string into a binary.

−x
−−discard−all
Delete all local symbols.

−X
−−discard−locals
Delete all temporary local symbols. (These symbols start with system-specific local label prefixes, typically **.L** for ELF systems or **L** for traditional a.out systems.)

−y *symbol*
−−trace−symbol=*symbol*
Print the name of each linked file in which *symbol* appears. This option may be given any number of times. On many systems it is necessary to prepend an underscore.

This option is useful when you have an undefined symbol in your link but don't know where the reference is coming from.

−Y *path*
Add *path* to the default library search path. This option exists for Solaris compatibility.

−z *keyword*
The recognized keywords are:

call−nop=prefix−addr
call−nop=suffix−nop
call−nop=prefix−*byte*
call−nop=suffix−*byte*
Specify the 1−byte `NOP` padding when transforming indirect call to a locally defined function, foo, via its GOT slot. **call−nop=prefix−addr** generates `0x67 call foo`. **call−nop=suffix−nop** generates `call foo 0x90`. **call−nop=prefix−***byte* generates *`byte`* `call foo`. **call−nop=suffix−***byte* generates `call foo` *`byte`*. Supported for i386 and x86_64.

cet−report=none
cet−report=warning
cet−report=error
Specify how to report the missing GNU_PROPERTY_X86_FEATURE_1_IBT and GNU_PROPERTY_X86_FEATURE_1_SHSTK properties in input .note.gnu.property section. **cet−report=none**, which is the default, will make the linker not report missing properties in input files. **cet−report=warning** will make the linker issue a warning for missing properties in input files. **cet−report=error** will make the linker issue an error for missing properties in input files. Note that **ibt** will turn off the missing GNU_PROPERTY_X86_FEATURE_1_IBT property report and **shstk** will turn off the missing GNU_PROPERTY_X86_FEATURE_1_SHSTK property report. Supported for Linux/i386 and Linux/x86_64.

combreloc
nocombreloc
Combine multiple dynamic relocation sections and sort to improve dynamic symbol lookup caching. Do not do this if **nocombreloc**.

common
nocommon
Generate common symbols with STT_COMMON type during a relocatable link. Use STT_OBJECT type if **nocommon**.

common−page−size=*value*
Set the page size most commonly used to *value*. Memory image layout will be optimized to minimize memory pages if the system is using pages of this size.

defs
Report unresolved symbol references from regular object files. This is done even if the linker is creating a non-symbolic shared library. This option is the inverse of **−z undefs**.

dynamic-undefined-weak
nodynamic-undefined-weak
Make undefined weak symbols dynamic when building a dynamic object, if they are referenced from a regular object file and not forced local by symbol visibility or versioning. Do not make them dynamic if **nodynamic-undefined-weak**. If neither option is given, a target may default to either option being in force, or make some other selection of undefined weak symbols dynamic. Not all targets support these options.

execstack
Marks the object as requiring executable stack.

global
This option is only meaningful when building a shared object. It makes the symbols defined by this shared object available for symbol resolution of subsequently loaded libraries.

globalaudit
This option is only meaningful when building a dynamic executable. This option marks the executable as requiring global auditing by setting the `DF_1_GLOBAUDIT` bit in the `DT_FLAGS_1` dynamic tag. Global auditing requires that any auditing library defined via the **−−depaudit** or **−P** command-line options be run for all dynamic objects loaded by the application.

ibtplt
Generate Intel Indirect Branch Tracking (IBT) enabled PLT entries. Supported for Linux/i386 and Linux/x86_64.

ibt Generate GNU_PROPERTY_X86_FEATURE_1_IBT in .note.gnu.property section to indicate compatibility with IBT. This also implies **ibtplt**. Supported for Linux/i386 and Linux/x86_64.

indirect-extern-access
noindirect-extern-access
Generate GNU_PROPERTY_1_NEEDED_INDIRECT_EXTERN_ACCESS in .note.gnu.property section to indicate that object file requires canonical function pointers and cannot be used with copy relocation. This option also implies **noextern-protected-data** and **nocopyreloc**. Supported for i386 and x86−64.

noindirect-extern-access removes GNU_PROPERTY_1_NEEDED_INDIRECT_EXTERN_ACCESS from .note.gnu.property section.

initfirst
This option is only meaningful when building a shared object. It marks the object so that its runtime initialization will occur before the runtime initialization of any other objects brought into the process at the same time. Similarly the runtime finalization of the object will occur after the runtime finalization of any other objects.

interpose
Specify that the dynamic loader should modify its symbol search order so that symbols in this shared library interpose all other shared libraries not so marked.

unique
nounique
When generating a shared library or other dynamically loadable ELF object mark it as one that should (by default) only ever be loaded once, and only in the main namespace (when using `dlmopen`). This is primarily used to mark fundamental libraries such as libc, libpthread et al which do not usually function correctly unless they are the sole instances of themselves. This behaviour can be overridden by the `dlmopen` caller and does not apply to certain loading mechanisms (such as audit libraries).

lam−u48
Generate GNU_PROPERTY_X86_FEATURE_1_LAM_U48 in .note.gnu.property section to indicate compatibility with Intel LAM_U48. Supported for Linux/x86_64.

lam−u57
Generate GNU_PROPERTY_X86_FEATURE_1_LAM_U57 in .note.gnu.property section to indicate compatibility with Intel LAM_U57. Supported for Linux/x86_64.

lam−u48−report=none
lam−u48−report=warning
lam−u48−report=error
Specify how to report the missing GNU_PROPERTY_X86_FEATURE_1_LAM_U48 property in input .note.gnu.property section. **lam−u48−report=none**, which is the default, will make the linker not report missing properties in input files. **lam−u48−report=warning** will make the linker issue a warning for missing properties in input files. **lam−u48−report=error** will make the linker issue an error for missing properties in input files. Supported for Linux/x86_64.

lam−u57−report=none
lam−u57−report=warning
lam−u57−report=error
Specify how to report the missing GNU_PROPERTY_X86_FEATURE_1_LAM_U57 property in input .note.gnu.property section. **lam−u57−report=none**, which is the default, will make the linker not report missing properties in input files. **lam−u57−report=warning** will make the linker issue a warning for missing properties in input files. **lam−u57−report=error** will make the linker issue an error for missing properties in input files. Supported for Linux/x86_64.

lam−report=none
lam−report=warning
lam−report=error
Specify how to report the missing GNU_PROPERTY_X86_FEATURE_1_LAM_U48 and GNU_PROPERTY_X86_FEATURE_1_LAM_U57 properties in input .note.gnu.property section. **lam−report=none**, which is the default, will make the linker not report missing properties in input files. **lam−report=warning** will make the linker issue a warning for missing properties in input files. **lam−report=error** will make the linker issue an error for missing properties in input files. Supported for Linux/x86_64.

lazy
When generating an executable or shared library, mark it to tell the dynamic linker to defer function call resolution to the point when the function is called (lazy binding), rather than at load time. Lazy binding is the default.

loadfltr
Specify that the object's filters be processed immediately at runtime.

max−page−size=*value*
Set the maximum memory page size supported to *value*.

mark-plt
nomark-plt
Mark PLT entries with dynamic tags, DT_X86_64_PLT, DT_X86_64_PLTSZ and DT_X86_64_PLTENT. Since this option stores a non-zero value in the r_addend field of

R_X86_64_JUMP_SLOT relocations, the resulting executables and shared libraries are incompatible with dynamic linkers, such as those in older versions of glibc without the change to ignore r_addend in R_X86_64_GLOB_DAT and R_X86_64_JUMP_SLOT relocations, which don't ignore the r_addend field of R_X86_64_JUMP_SLOT relocations. Supported for x86_64.

muldefs
Allow multiple definitions.

nocopyreloc
Disable linker generated .dynbss variables used in place of variables defined in shared libraries. May result in dynamic text relocations.

nodefaultlib
Specify that the dynamic loader search for dependencies of this object should ignore any default library search paths.

nodelete
Specify that the object shouldn't be unloaded at runtime.

nodlopen
Specify that the object is not available to `dlopen`.

nodump
Specify that the object can not be dumped by `dldump`.

noexecstack
Marks the object as not requiring executable stack.

noextern-protected-data
Don't treat protected data symbols as external when building a shared library. This option overrides the linker backend default. It can be used to work around incorrect relocations against protected data symbols generated by compiler. Updates on protected data symbols by another module aren't visible to the resulting shared library. Supported for i386 and x86−64.

noreloc-overflow
Disable relocation overflow check. This can be used to disable relocation overflow check if there will be no dynamic relocation overflow at run-time. Supported for x86_64.

now
When generating an executable or shared library, mark it to tell the dynamic linker to resolve all symbols when the program is started, or when the shared library is loaded by dlopen, instead of deferring function call resolution to the point when the function is first called.

origin
Specify that the object requires **`$ORIGIN`** handling in paths.

pack-relative-relocs
nopack-relative-relocs
Generate compact relative relocation in position-independent executable and shared library. It adds `DT_RELR`, `DT_RELRSZ` and `DT_RELRENT` entries to the dynamic section. It is ignored when building position-dependent executable and relocatable output. **nopack-relative-relocs** is the default, which disables compact relative relocation. When linked against the GNU C Library, a GLIBC_ABI_DT_RELR symbol version dependency on the shared C Library is added to the output. Supported for i386 and x86−64.

relro
norelro
Create an ELF `PT_GNU_RELRO` segment header in the object. This specifies a memory segment that should be made read-only after relocation, if supported. Specifying **common-page-size** smaller than the system page size will render this protection ineffective. Don't create an ELF `PT_GNU_RELRO` segment if **norelro**.

report-relative-reloc
Report dynamic relative relocations generated by linker. Supported for Linux/i386 and Linux/x86_64.

sectionheader
nosectionheader
Generate section header. Don't generate section header if **nosectionheader** is used. **sectionheader** is the default.

separate-code
noseparate-code
Create separate code `PT_LOAD` segment header in the object. This specifies a memory segment that should contain only instructions and must be in wholly disjoint pages from any other data. Don't create separate code `PT_LOAD` segment if **noseparate-code** is used.

shstk
Generate GNU_PROPERTY_X86_FEATURE_1_SHSTK in .note.gnu.property section to indicate compatibility with Intel Shadow Stack. Supported for Linux/i386 and Linux/x86_64.

stack−size=*value*
Specify a stack size for an ELF `PT_GNU_STACK` segment. Specifying zero will override any default non-zero sized `PT_GNU_STACK` segment creation.

start-stop-gc
nostart-stop-gc
When **−−gc−sections** is in effect, a reference from a retained section to `__start_SECNAME` or `__stop_SECNAME` causes all input sections named `SECNAME` to also be retained, if `SECNAME` is representable as a C identifier and either `__start_SECNAME` or `__stop_SECNAME` is synthesized by the linker. **−z start-stop-gc** disables this effect, allowing sections to be garbage collected as if the special synthesized symbols were not defined. **−z start-stop-gc** has no effect on a definition of `__start_SECNAME` or `__stop_SECNAME` in an object file or linker script. Such a definition will prevent the linker providing a synthesized `__start_SECNAME` or `__stop_SECNAME` respectively, and therefore the special treatment by garbage collection for those references.

start−stop−visibility=*value*
Specify the ELF symbol visibility for synthesized `__start_SECNAME` and `__stop_SECNAME` symbols. *value* must be exactly **default**, **internal**, **hidden**, or **protected**. If no **−z start-stop-visibility** option is given, **protected** is used for compatibility with historical practice. However, it's highly recommended to use **−z start−stop−visibility=hidden** in new programs and shared libraries so that these symbols are not exported between shared objects, which is not usually what's intended.

text
notext
textoff
Report an error if DT_TEXTREL is set, i.e., if the position-independent or shared object has dynamic relocations in read-only sections. Don't report an error if **notext** or **textoff**.

undefs
Do not report unresolved symbol references from regular object files, either when creating an executable, or when creating a shared library. This option is the inverse of **−z defs**.

unique-symbol
nounique-symbol
Avoid duplicated local symbol names in the symbol string table. Append ".`number`" to duplicated local symbol names if **unique-symbol** is used. **nounique-symbol** is the default.

x86−64−baseline
x86−64−v2
x86−64−v3
x86−64−v4
Specify the x86−64 ISA level needed in .note.gnu.property section. **x86−64−baseline** generates `GNU_PROPERTY_X86_ISA_1_BASELINE`. **x86−64−v2** generates `GNU_PROPERTY_X86_ISA_1_V2`. **x86−64−v3** generates `GNU_PROPERTY_X86_ISA_1_V3`. **x86−64−v4** generates `GNU_PROPERTY_X86_ISA_1_V4`. Supported for Linux/i386 and Linux/x86_64.

Other keywords are ignored for Solaris compatibility.

−(*archives* **−)**
−−start−group *archives* **−−end−group**
The *archives* should be a list of archive files. They may be either explicit file names, or **−l** options.

The specified archives are searched repeatedly until no new undefined references are created. Normally, an archive is searched only once in the order that it is specified on the command line. If a symbol in that archive is needed to resolve an undefined symbol referred to by an object in an archive that appears later on the command line, the linker would not be able to resolve that reference. By grouping the archives, they will all be searched repeatedly until all possible references are resolved.

Using this option has a significant performance cost. It is best to use it only when there are unavoidable circular references between two or more archives.

−−accept−unknown−input−arch
−−no−accept−unknown−input−arch
Tells the linker to accept input files whose architecture cannot be recognised. The assumption is that the user knows what they are doing and deliberately wants to link in these unknown input files. This was the default behaviour of the linker, before release 2.14. The default behaviour from release 2.14 onwards is to reject such input files, and so the **−−accept−unknown−input−arch** option has been added to restore the old behaviour.

−−as−needed
−−no−as−needed
This option affects ELF DT_NEEDED tags for dynamic libraries mentioned on the command line after the **−−as−needed** option. Normally the linker will add a DT_NEEDED tag for each dynamic library mentioned on the command line, regardless of whether the library is actually needed or not. **−−as−needed** causes a DT_NEEDED tag to only be emitted for a library that *at that point in the link* satisfies a non-weak undefined symbol reference from a regular object file or, if the library is not found in the DT_NEEDED lists of other needed libraries, a non-weak undefined symbol reference from another needed dynamic library. Object files or libraries appearing on the command line *after* the library in question do not affect whether the library is seen as needed. This is similar to the rules for extraction of object files from archives. **−−no−as−needed** restores the default behaviour.

Note: On Linux based systems the **−−as−needed** option also has an affect on the behaviour of the **−−rpath** and **−−rpath−link** options. See the description of **−−rpath−link** for more details.

−−add−needed
−−no−add−needed
These two options have been deprecated because of the similarity of their names to the **−−as−needed** and **−−no−as−needed** options. They have been replaced by **−−copy−dt−needed−entries** and **−−no−copy−dt−needed−entries**.

−assert *keyword*
This option is ignored for SunOS compatibility.

−Bdynamic

−dy
−call_shared
Link against dynamic libraries. This is only meaningful on platforms for which shared libraries are supported. This option is normally the default on such platforms. The different variants of this option are for compatibility with various systems. You may use this option multiple times on the command line: it affects library searching for **−l** options which follow it.

−Bgroup
Set the `DF_1_GROUP` flag in the `DT_FLAGS_1` entry in the dynamic section. This causes the runtime linker to handle lookups in this object and its dependencies to be performed only inside the group. **−−unresolved−symbols=report−all** is implied. This option is only meaningful on ELF platforms which support shared libraries.

−Bstatic
−dn
−non_shared
−static
Do not link against shared libraries. This is only meaningful on platforms for which shared libraries are supported. The different variants of this option are for compatibility with various systems. You may use this option multiple times on the command line: it affects library searching for **−l** options which follow it. This option also implies **−−unresolved−symbols=report−all**. This option can be used with **−shared**. Doing so means that a shared library is being created but that all of the library's external references must be resolved by pulling in entries from static libraries.

−Bsymbolic
When creating a shared library, bind references to global symbols to the definition within the shared library, if any. Normally, it is possible for a program linked against a shared library to override the definition within the shared library. This option is only meaningful on ELF platforms which support shared libraries.

−Bsymbolic−functions
When creating a shared library, bind references to global function symbols to the definition within the shared library, if any. This option is only meaningful on ELF platforms which support shared libraries.

−Bno−symbolic
This option can cancel previously specified **−Bsymbolic** and **−Bsymbolic−functions**.

−−dynamic−list=*dynamic-list-file*
Specify the name of a dynamic list file to the linker. This is typically used when creating shared libraries to specify a list of global symbols whose references shouldn't be bound to the definition within the shared library, or creating dynamically linked executables to specify a list of symbols which should be added to the symbol table in the executable. This option is only meaningful on ELF platforms which support shared libraries.

The format of the dynamic list is the same as the version node without scope and node name. See **VERSION** for more information.

−−dynamic−list−data
Include all global data symbols to the dynamic list.

−−dynamic−list−cpp−new
Provide the builtin dynamic list for C++ operator new and delete. It is mainly useful for building shared libstdc++.

−−dynamic−list−cpp−typeinfo
Provide the builtin dynamic list for C++ runtime type identification.

−−check−sections

−−no−check−sections
Asks the linker *not* to check section addresses after they have been assigned to see if there are any overlaps. Normally the linker will perform this check, and if it finds any overlaps it will produce suitable error messages. The linker does know about, and does make allowances for sections in overlays. The default behaviour can be restored by using the command-line switch **−−check−sections**. Section overlap is not usually checked for relocatable links. You can force checking in that case by using the **−−check−sections** option.

−−copy−dt−needed−entries
−−no−copy−dt−needed−entries
This option affects the treatment of dynamic libraries referred to by DT_NEEDED tags *inside* ELF dynamic libraries mentioned on the command line. Normally the linker won't add a DT_NEEDED tag to the output binary for each library mentioned in a DT_NEEDED tag in an input dynamic library. With **−−copy−dt−needed−entries** specified on the command line however any dynamic libraries that follow it will have their DT_NEEDED entries added. The default behaviour can be restored with **−−no−copy−dt−needed−entries**.

This option also has an effect on the resolution of symbols in dynamic libraries. With **−−copy−dt−needed−entries** dynamic libraries mentioned on the command line will be recursively searched, following their DT_NEEDED tags to other libraries, in order to resolve symbols required by the output binary. With the default setting however the searching of dynamic libraries that follow it will stop with the dynamic library itself. No DT_NEEDED links will be traversed to resolve symbols.

−−cref
Output a cross reference table. If a linker map file is being generated, the cross reference table is printed to the map file. Otherwise, it is printed on the standard output.

The format of the table is intentionally simple, so that it may be easily processed by a script if necessary. The symbols are printed out, sorted by name. For each symbol, a list of file names is given. If the symbol is defined, the first file listed is the location of the definition. If the symbol is defined as a common value then any files where this happens appear next. Finally any files that reference the symbol are listed.

−−ctf−variables
−−no−ctf−variables
The CTF debuginfo format supports a section which encodes the names and types of variables found in the program which do not appear in any symbol table. These variables clearly cannot be looked up by address by conventional debuggers, so the space used for their types and names is usually wasted: the types are usually small but the names are often not. **−−ctf−variables** causes the generation of such a section. The default behaviour can be restored with **−−no−ctf−variables**.

−−ctf−share−types=*method*
Adjust the method used to share types between translation units in CTF.

share-unconflicted
Put all types that do not have ambiguous definitions into the shared dictionary, where debuggers can easily access them, even if they only occur in one translation unit. This is the default.

share-duplicated
Put only types that occur in multiple translation units into the shared dictionary: types with only one definition go into per-translation-unit dictionaries. Types with ambiguous definitions in multiple translation units always go into per-translation-unit dictionaries. This tends to make the CTF larger, but may reduce the amount of CTF in the shared dictionary. For very large projects this may speed up opening the CTF and save memory in the CTF consumer at runtime.

−−no−define−common
This option inhibits the assignment of addresses to common symbols. The script command `INHIBIT_COMMON_ALLOCATION` has the same effect.

The **−−no−define−common** option allows decoupling the decision to assign addresses to Common

symbols from the choice of the output file type; otherwise a non-Relocatable output type forces assigning addresses to Common symbols. Using **--no-define-common** allows Common symbols that are referenced from a shared library to be assigned addresses only in the main program. This eliminates the unused duplicate space in the shared library, and also prevents any possible confusion over resolving to the wrong duplicate when there are many dynamic modules with specialized search paths for runtime symbol resolution.

--force-group-allocation
This option causes the linker to place section group members like normal input sections, and to delete the section groups. This is the default behaviour for a final link but this option can be used to change the behaviour of a relocatable link (**-r**). The script command `FORCE_GROUP_ALLOCATION` has the same effect.

--defsym=*symbol***=***expression*
Create a global symbol in the output file, containing the absolute address given by *expression*. You may use this option as many times as necessary to define multiple symbols in the command line. A limited form of arithmetic is supported for the *expression* in this context: you may give a hexadecimal constant or the name of an existing symbol, or use + and - to add or subtract hexadecimal constants or symbols. If you need more elaborate expressions, consider using the linker command language from a script. *Note:* there should be no white space between *symbol*, the equals sign ("**=**"), and *expression*.

The linker processes **--defsym** arguments and **-T** arguments in order, placing **--defsym** before **-T** will define the symbol before the linker script from **-T** is processed, while placing **--defsym** after **-T** will define the symbol after the linker script has been processed. This difference has consequences for expressions within the linker script that use the **--defsym** symbols, which order is correct will depend on what you are trying to achieve.

--demangle[=*style***]**
--no-demangle
These options control whether to demangle symbol names in error messages and other output. When the linker is told to demangle, it tries to present symbol names in a readable fashion: it strips leading underscores if they are used by the object file format, and converts C++ mangled symbol names into user readable names. Different compilers have different mangling styles. The optional demangling style argument can be used to choose an appropriate demangling style for your compiler. The linker will demangle by default unless the environment variable **COLLECT_NO_DEMANGLE** is set. These options may be used to override the default.

-I*file*
--dynamic-linker=*file*
Set the name of the dynamic linker. This is only meaningful when generating dynamically linked ELF executables. The default dynamic linker is normally correct; don't use this unless you know what you are doing.

--no-dynamic-linker
When producing an executable file, omit the request for a dynamic linker to be used at load-time. This is only meaningful for ELF executables that contain dynamic relocations, and usually requires entry point code that is capable of processing these relocations.

--embedded-relocs
This option is similar to the **--emit-relocs** option except that the relocs are stored in a target-specific section. This option is only supported by the **BFIN**, **CR16** and *M68K* targets.

--disable-multiple-abs-defs
Do not allow multiple definitions with symbols included in filename invoked by -R or --just-symbols

--fatal-warnings
--no-fatal-warnings
Treat all warnings as errors. The default behaviour can be restored with the option **--no-fatal-warnings**.

−w
−−no−warnings
Do not display any warning or error messages. This overrides **−−fatal−warnings** if it has been enabled. This option can be used when it is known that the output binary will not work, but there is still a need to create it.

−−force−exe−suffix
Make sure that an output file has a .exe suffix.

If a successfully built fully linked output file does not have a `.exe` or `.dll` suffix, this option forces the linker to copy the output file to one of the same name with a `.exe` suffix. This option is useful when using unmodified Unix makefiles on a Microsoft Windows host, since some versions of Windows won't run an image unless it ends in a `.exe` suffix.

−−gc−sections
−−no−gc−sections
Enable garbage collection of unused input sections. It is ignored on targets that do not support this option. The default behaviour (of not performing this garbage collection) can be restored by specifying **−−no−gc−sections** on the command line. Note that garbage collection for COFF and PE format targets is supported, but the implementation is currently considered to be experimental.

−−gc−sections decides which input sections are used by examining symbols and relocations. The section containing the entry symbol and all sections containing symbols undefined on the command-line will be kept, as will sections containing symbols referenced by dynamic objects. Note that when building shared libraries, the linker must assume that any visible symbol is referenced. Once this initial set of sections has been determined, the linker recursively marks as used any section referenced by their relocations. See **−−entry**, **−−undefined**, and **−−gc−keep−exported**.

This option can be set when doing a partial link (enabled with option **−r**). In this case the root of symbols kept must be explicitly specified either by one of the options **−−entry**, **−−undefined**, or **−−gc−keep−exported** or by a `ENTRY` command in the linker script.

As a GNU extension, ELF input sections marked with the `SHF_GNU_RETAIN` flag will not be garbage collected.

−−print−gc−sections
−−no−print−gc−sections
List all sections removed by garbage collection. The listing is printed on stderr. This option is only effective if garbage collection has been enabled via the **−−gc−sections**) option. The default behaviour (of not listing the sections that are removed) can be restored by specifying **−−no−print−gc−sections** on the command line.

−−gc−keep−exported
When **−−gc−sections** is enabled, this option prevents garbage collection of unused input sections that contain global symbols having default or protected visibility. This option is intended to be used for executables where unreferenced sections would otherwise be garbage collected regardless of the external visibility of contained symbols. Note that this option has no effect when linking shared objects since it is already the default behaviour. This option is only supported for ELF format targets.

−−print−output−format
Print the name of the default output format (perhaps influenced by other command-line options). This is the string that would appear in an `OUTPUT_FORMAT` linker script command.

−−print−memory−usage
Print used size, total size and used size of memory regions created with the **MEMORY** command. This is useful on embedded targets to have a quick view of amount of free memory. The format of the output has one headline and one line per region. It is both human readable and easily parsable by tools. Here is an example of an output:

```
Memory region         Used Size  Region Size  %age Used
             ROM:         256 KB         1 MB     25.00%
             RAM:           32 B         2 GB      0.00%
```

--help
Print a summary of the command-line options on the standard output and exit.

--target-help
Print a summary of all target-specific options on the standard output and exit.

-Map=*mapfile*
Print a link map to the file *mapfile*. See the description of the **-M** option, above. If *mapfile* is just the character - then the map will be written to stdout.

Specifying a directory as *mapfile* causes the linker map to be written as a file inside the directory. Normally name of the file inside the directory is computed as the basename of the *output* file with `.map` appended. If however the special character `%` is used then this will be replaced by the full path of the output file. Additionally if there are any characters after the % symbol then `.map` will no longer be appended.

```
-o foo.exe -Map=bar                  [Creates ./bar]
-o ../dir/foo.exe -Map=bar           [Creates ./bar]
-o foo.exe -Map=../dir               [Creates ../dir/foo.exe.map]
-o ../dir2/foo.exe -Map=../dir       [Creates ../dir/foo.exe.map]
-o foo.exe -Map=%                    [Creates ./foo.exe.map]
-o ../dir/foo.exe -Map=%             [Creates ../dir/foo.exe.map]
-o foo.exe -Map=%.bar                [Creates ./foo.exe.bar]
-o ../dir/foo.exe -Map=%.bar         [Creates ../dir/foo.exe.bar]
-o ../dir2/foo.exe -Map=../dir/%     [Creates ../dir/../dir2/foo.exe.map
-o ../dir2/foo.exe -Map=../dir/%.bar [Creates ../dir/../dir2/foo.exe.bar
```

It is an error to specify more than one `%` character.

If the map file already exists then it will be overwritten by this operation.

--no-keep-memory
ld normally optimizes for speed over memory usage by caching the symbol tables of input files in memory. This option tells **ld** to instead optimize for memory usage, by rereading the symbol tables as necessary. This may be required if **ld** runs out of memory space while linking a large executable.

--no-undefined
-z defs
Report unresolved symbol references from regular object files. This is done even if the linker is creating a non-symbolic shared library. The switch **--[no-]allow-shlib-undefined** controls the behaviour for reporting unresolved references found in shared libraries being linked in.

The effects of this option can be reverted by using `-z undefs`.

--allow-multiple-definition
-z muldefs
Normally when a symbol is defined multiple times, the linker will report a fatal error. These options allow multiple definitions and the first definition will be used.

--allow-shlib-undefined
--no-allow-shlib-undefined
Allows or disallows undefined symbols in shared libraries. This switch is similar to **--no-undefined** except that it determines the behaviour when the undefined symbols are in a shared library rather than a regular object file. It does not affect how undefined symbols in regular object files are handled.

The default behaviour is to report errors for any undefined symbols referenced in shared libraries if the linker is being used to create an executable, but to allow them if the linker is being used to create a shared library.

The reasons for allowing undefined symbol references in shared libraries specified at link time are that:

- A shared library specified at link time may not be the same as the one that is available at load time, so the symbol might actually be resolvable at load time.
- There are some operating systems, eg BeOS and HPPA, where undefined symbols in shared libraries are normal.

  The BeOS kernel for example patches shared libraries at load time to select whichever function is most appropriate for the current architecture. This is used, for example, to dynamically select an appropriate memset function.

−−error−handling−script=*scriptname*
If this option is provided then the linker will invoke *scriptname* whenever an error is encountered. Currently however only two kinds of error are supported: missing symbols and missing libraries. Two arguments will be passed to script: the keyword "undefined-symbol" or ‘missing−lib" and the *name* of the undefined symbol or missing library. The intention is that the script will provide suggestions to the user as to where the symbol or library might be found. After the script has finished then the normal linker error message will be displayed.

The availability of this option is controlled by a configure time switch, so it may not be present in specific implementations.

−−no−undefined−version
Normally when a symbol has an undefined version, the linker will ignore it. This option disallows symbols with undefined version and a fatal error will be issued instead.

−−default−symver
Create and use a default symbol version (the soname) for unversioned exported symbols.

−−default−imported−symver
Create and use a default symbol version (the soname) for unversioned imported symbols.

−−no−warn−mismatch
Normally **ld** will give an error if you try to link together input files that are mismatched for some reason, perhaps because they have been compiled for different processors or for different endiannesses. This option tells **ld** that it should silently permit such possible errors. This option should only be used with care, in cases when you have taken some special action that ensures that the linker errors are inappropriate.

−−no−warn−search−mismatch
Normally **ld** will give a warning if it finds an incompatible library during a library search. This option silences the warning.

−−no−whole−archive
Turn off the effect of the **−−whole−archive** option for subsequent archive files.

−−noinhibit−exec
Retain the executable output file whenever it is still usable. Normally, the linker will not produce an output file if it encounters errors during the link process; it exits without writing an output file when it issues any error whatsoever.

−nostdlib
Only search library directories explicitly specified on the command line. Library directories specified in linker scripts (including linker scripts specified on the command line) are ignored.

−−oformat=*output-format*
ld may be configured to support more than one kind of object file. If your **ld** is configured this way, you can use the **−−oformat** option to specify the binary format for the output object file. Even when **ld** is configured to support alternative object formats, you don’t usually need to specify this, as **ld** should be configured to produce as a default output format the most usual format on each machine. *output-format* is a text string, the name of a particular format supported by the BFD libraries. (You

can list the available binary formats with **objdump −i**.) The script command `OUTPUT_FORMAT` can also specify the output format, but this option overrides it.

−−out−implib *file*
Create an import library in *file* corresponding to the executable the linker is generating (eg. a DLL or ELF program). This import library (which should be called `*.dll.a` or `*.a` for DLLs) may be used to link clients against the generated executable; this behaviour makes it possible to skip a separate import library creation step (eg. `dlltool` for DLLs). This option is only available for the i386 PE and ELF targetted ports of the linker.

−pie
−−pic−executable
Create a position independent executable. This is currently only supported on ELF platforms. Position independent executables are similar to shared libraries in that they are relocated by the dynamic linker to the virtual address the OS chooses for them (which can vary between invocations). Like normal dynamically linked executables they can be executed and symbols defined in the executable cannot be overridden by shared libraries.

−no−pie
Create a position dependent executable. This is the default.

−qmagic
This option is ignored for Linux compatibility.

−Qy
This option is ignored for SVR4 compatibility.

−−relax
−−no−relax
An option with machine dependent effects. This option is only supported on a few targets.

On some platforms the **−−relax** option performs target specific, global optimizations that become possible when the linker resolves addressing in the program, such as relaxing address modes, synthesizing new instructions, selecting shorter version of current instructions, and combining constant values.

On some platforms these link time global optimizations may make symbolic debugging of the resulting executable impossible. This is known to be the case for the Matsushita MN10200 and MN10300 family of processors.

On platforms where the feature is supported, the option **−−no−relax** will disable it.

On platforms where the feature is not supported, both **−−relax** and **−−no−relax** are accepted, but ignored.

−−retain−symbols−file=*filename*
Retain *only* the symbols listed in the file *filename*, discarding all others. *filename* is simply a flat file, with one symbol name per line. This option is especially useful in environments (such as VxWorks) where a large global symbol table is accumulated gradually, to conserve run-time memory.

−−retain−symbols−file does *not* discard undefined symbols, or symbols needed for relocations.

You may only specify **−−retain−symbols−file** once in the command line. It overrides **−s** and **−S**.

−rpath=*dir*
Add a directory to the runtime library search path. This is used when linking an ELF executable with shared objects. All **−rpath** arguments are concatenated and passed to the runtime linker, which uses them to locate shared objects at runtime.

The **−rpath** option is also used when locating shared objects which are needed by shared objects explicitly included in the link; see the description of the **−rpath−link** option. Searching **−rpath** in this way is only supported by native linkers and cross linkers which have been configured with the **−−with−sysroot** option.

If **−rpath** is not used when linking an ELF executable, the contents of the environment variable `LD_RUN_PATH` will be used if it is defined.

The **−rpath** option may also be used on SunOS. By default, on SunOS, the linker will form a runtime search path out of all the **−L** options it is given. If a **−rpath** option is used, the runtime search path will be formed exclusively using the **−rpath** options, ignoring the **−L** options. This can be useful when using gcc, which adds many **−L** options which may be on NFS mounted file systems.

For compatibility with other ELF linkers, if the **−R** option is followed by a directory name, rather than a file name, it is treated as the **−rpath** option.

−rpath−link=*dir*

When using ELF or SunOS, one shared library may require another. This happens when an `ld −shared` link includes a shared library as one of the input files.

When the linker encounters such a dependency when doing a non-shared, non-relocatable link, it will automatically try to locate the required shared library and include it in the link, if it is not included explicitly. In such a case, the **−rpath−link** option specifies the first set of directories to search. The **−rpath−link** option may specify a sequence of directory names either by specifying a list of names separated by colons, or by appearing multiple times.

The tokens *$ORIGIN* and *$LIB* can appear in these search directories. They will be replaced by the full path to the directory containing the program or shared object in the case of *$ORIGIN* and either **lib** − for 32−bit binaries − or **lib64** − for 64−bit binaries − in the case of *$LIB*.

The alternative form of these tokens − *${ORIGIN}* and *${LIB}* can also be used. The token *$PLATFORM* is not supported.

This option should be used with caution as it overrides the search path that may have been hard compiled into a shared library. In such a case it is possible to use unintentionally a different search path than the runtime linker would do.

The linker uses the following search paths to locate required shared libraries:

1. Any directories specified by **−rpath−link** options.
2. Any directories specified by **−rpath** options. The difference between **−rpath** and **−rpath−link** is that directories specified by **−rpath** options are included in the executable and used at runtime, whereas the **−rpath−link** option is only effective at link time. Searching **−rpath** in this way is only supported by native linkers and cross linkers which have been configured with the **−−with−sysroot** option.
3. On an ELF system, for native linkers, if the **−rpath** and **−rpath−link** options were not used, search the contents of the environment variable `LD_RUN_PATH`.
4. On SunOS, if the **−rpath** option was not used, search any directories specified using **−L** options.
5. For a native linker, search the contents of the environment variable `LD_LIBRARY_PATH`.
6. For a native ELF linker, the directories in `DT_RUNPATH` or `DT_RPATH` of a shared library are searched for shared libraries needed by it. The `DT_RPATH` entries are ignored if `DT_RUNPATH` entries exist.
7. For a linker for a Linux system, if the file */etc/ld.so.conf* exists, the list of directories found in that file. Note: the path to this file is prefixed with the `sysroot` value, if that is defined, and then any `prefix` string if the linker was configured with the **−−prefix=<path>** option.
8. For a native linker on a FreeBSD system, any directories specified by the `_PATH_ELF_HINTS` macro defined in the *elf−hints.h* header file.
9. Any directories specified by a `SEARCH_DIR` command in a linker script given on the command line, including scripts specified by **−T** (but not **−dT**).

10. The default directories, normally */lib* and */usr/lib*.
11. Any directories specified by a plugin LDPT_SET_EXTRA_LIBRARY_PATH.
12. Any directories specified by a `SEARCH_DIR` command in a default linker script.

Note however on Linux based systems there is an additional caveat: If the **−−as−needed** option is active *and* a shared library is located which would normally satisfy the search *and* this library does not have DT_NEEDED tag for *libc.so and* there is a shared library later on in the set of search directories which also satisfies the search *and* this second shared library does have a DT_NEEDED tag for *libc.so then* the second library will be selected instead of the first.

If the required shared library is not found, the linker will issue a warning and continue with the link.

−shared
−Bshareable
Create a shared library. This is currently only supported on ELF, XCOFF and SunOS platforms. On SunOS, the linker will automatically create a shared library if the **−e** option is not used and there are undefined symbols in the link.

−−sort−common
−−sort−common=ascending
−−sort−common=descending
This option tells **ld** to sort the common symbols by alignment in ascending or descending order when it places them in the appropriate output sections. The symbol alignments considered are sixteen-byte or larger, eight-byte, four-byte, two-byte, and one-byte. This is to prevent gaps between symbols due to alignment constraints. If no sorting order is specified, then descending order is assumed.

−−sort−section=name
This option will apply `SORT_BY_NAME` to all wildcard section patterns in the linker script.

−−sort−section=alignment
This option will apply `SORT_BY_ALIGNMENT` to all wildcard section patterns in the linker script.

−−spare−dynamic−tags=*count*
This option specifies the number of empty slots to leave in the .dynamic section of ELF shared objects. Empty slots may be needed by post processing tools, such as the prelinker. The default is 5.

−−split−by−file[=*size***]**
Similar to **−−split−by−reloc** but creates a new output section for each input file when *size* is reached. *size* defaults to a size of 1 if not given.

−−split−by−reloc[=*count***]**
Tries to creates extra sections in the output file so that no single output section in the file contains more than *count* relocations. This is useful when generating huge relocatable files for downloading into certain real time kernels with the COFF object file format; since COFF cannot represent more than 65535 relocations in a single section. Note that this will fail to work with object file formats which do not support arbitrary sections. The linker will not split up individual input sections for redistribution, so if a single input section contains more than *count* relocations one output section will contain that many relocations. *count* defaults to a value of 32768.

−−stats
Compute and display statistics about the operation of the linker, such as execution time and memory usage.

−−sysroot=*directory*
Use *directory* as the location of the sysroot, overriding the configure-time default. This option is only supported by linkers that were configured using **−−with−sysroot**.

−−task−link
This is used by COFF/PE based targets to create a task-linked object file where all of the global symbols have been converted to statics.

−−traditional−format
For some targets, the output of **ld** is different in some ways from the output of some existing linker. This switch requests **ld** to use the traditional format instead.

For example, on SunOS, **ld** combines duplicate entries in the symbol string table. This can reduce the size of an output file with full debugging information by over 30 percent. Unfortunately, the SunOS `dbx` program can not read the resulting program (`gdb` has no trouble). The **−−traditional−format** switch tells **ld** to not combine duplicate entries.

−−section−start=*sectionname***=***org*
Locate a section in the output file at the absolute address given by *org*. You may use this option as many times as necessary to locate multiple sections in the command line. *org* must be a single hexadecimal integer; for compatibility with other linkers, you may omit the leading **0x** usually associated with hexadecimal values. *Note:* there should be no white space between *sectionname*, the equals sign ("**=**"), and *org*.

−Tbss=*org*
−Tdata=*org*
−Ttext=*org*
Same as **−−section−start**, with `.bss`, `.data` or `.text` as the *sectionname*.

−Ttext−segment=*org*
When creating an ELF executable, it will set the address of the first byte of the text segment.

−Trodata−segment=*org*
When creating an ELF executable or shared object for a target where the read-only data is in its own segment separate from the executable text, it will set the address of the first byte of the read-only data segment.

−Tldata−segment=*org*
When creating an ELF executable or shared object for x86−64 medium memory model, it will set the address of the first byte of the ldata segment.

−−unresolved−symbols=*method*
Determine how to handle unresolved symbols. There are four possible values for **method**:

ignore-all
Do not report any unresolved symbols.

report-all
Report all unresolved symbols. This is the default.

ignore-in-object-files
Report unresolved symbols that are contained in shared libraries, but ignore them if they come from regular object files.

ignore-in-shared-libs
Report unresolved symbols that come from regular object files, but ignore them if they come from shared libraries. This can be useful when creating a dynamic binary and it is known that all the shared libraries that it should be referencing are included on the linker's command line.

The behaviour for shared libraries on their own can also be controlled by the **−−[no−]allow−shlib−undefined** option.

Normally the linker will generate an error message for each reported unresolved symbol but the option **−−warn−unresolved−symbols** can change this to a warning.

−−dll−verbose
−−verbose[=*NUMBER***]**
Display the version number for **ld** and list the linker emulations supported. Display which input files can and cannot be opened. Display the linker script being used by the linker. If the optional *NUMBER* argument > 1, plugin symbol status will also be displayed.

−−version−script=*version-scriptfile*
Specify the name of a version script to the linker. This is typically used when creating shared libraries to specify additional information about the version hierarchy for the library being created. This option is only fully supported on ELF platforms which support shared libraries; see **VERSION**. It is partially supported on PE platforms, which can use version scripts to filter symbol visibility in auto-export mode: any symbols marked **local** in the version script will not be exported.

−−warn−common
Warn when a common symbol is combined with another common symbol or with a symbol definition. Unix linkers allow this somewhat sloppy practice, but linkers on some other operating systems do not. This option allows you to find potential problems from combining global symbols. Unfortunately, some C libraries use this practice, so you may get some warnings about symbols in the libraries as well as in your programs.

There are three kinds of global symbols, illustrated here by C examples:

int i = 1;
A definition, which goes in the initialized data section of the output file.

extern int i;
An undefined reference, which does not allocate space. There must be either a definition or a common symbol for the variable somewhere.

int i;
A common symbol. If there are only (one or more) common symbols for a variable, it goes in the uninitialized data area of the output file. The linker merges multiple common symbols for the same variable into a single symbol. If they are of different sizes, it picks the largest size. The linker turns a common symbol into a declaration, if there is a definition of the same variable.

The **−−warn−common** option can produce five kinds of warnings. Each warning consists of a pair of lines: the first describes the symbol just encountered, and the second describes the previous symbol encountered with the same name. One or both of the two symbols will be a common symbol.

1. Turning a common symbol into a reference, because there is already a definition for the symbol.

```
<file>(<section>): warning: common of `<symbol>'
   overridden by definition
<file>(<section>): warning: defined here
```

2. Turning a common symbol into a reference, because a later definition for the symbol is encountered. This is the same as the previous case, except that the symbols are encountered in a different order.

```
<file>(<section>): warning: definition of `<symbol>'
   overriding common
<file>(<section>): warning: common is here
```

3. Merging a common symbol with a previous same-sized common symbol.

```
<file>(<section>): warning: multiple common
   of `<symbol>'
<file>(<section>): warning: previous common is here
```

4. Merging a common symbol with a previous larger common symbol.

```
<file>(<section>): warning: common of `<symbol>'
   overridden by larger common
<file>(<section>): warning: larger common is here
```

5. Merging a common symbol with a previous smaller common symbol. This is the same as the previous case, except that the symbols are encountered in a different order.

```
<file>(<section>): warning: common of `<symbol>'
   overriding smaller common
<file>(<section>): warning: smaller common is here
```

−−warn−constructors
Warn if any global constructors are used. This is only useful for a few object file formats. For formats like COFF or ELF, the linker can not detect the use of global constructors.

−−warn−execstack
−−warn−execstack−objects
−−no−warn−execstack
On ELF platforms the linker may generate warning messages if it is asked to create an output file that contains an executable stack. There are three possible states:

1. Do not generate any warnings.
2. Always generate warnings, even if the executable stack is requested via the **−z execstack** command line option.
3. Only generate a warning if an object file requests an executable stack, but not if the **−z execstack** option is used.

The default state depends upon how the linker was configured when it was built. The **−−no−warn−execstack** option always puts the linker into the no-warnings state. The **−−warn−execstack** option puts the linker into the warn-always state. The **−−warn−execstack−objects** option puts the linker into the warn-for-object-files-only state.

Note: ELF format input files can specify that they need an executable stack by having a *.note.GNU−stack* section with the executable bit set in its section flags. They can specify that they do not need an executable stack by having the same section, but without the executable flag bit set. If an input file does not have a *.note.GNU−stack* section then the default behaviour is target specific. For some targets, then absence of such a section implies that an executable stack *is* required. This is often a problem for hand crafted assembler files.

−−error−execstack
−−no−error−execstack
If the linker is going to generate a warning message about an executable stack then the **−−error−execstack** option will instead change that warning into an error. Note − this option does not change the linker's execstack warning generation state. Use **−−warn−execstack** or **−−warn−execstack−objects** to set a specific warning state.

The **−−no−error−execstack** option will restore the default behaviour of generating warning messages.

−−warn−multiple−gp
Warn if multiple global pointer values are required in the output file. This is only meaningful for certain processors, such as the Alpha. Specifically, some processors put large-valued constants in a special section. A special register (the global pointer) points into the middle of this section, so that constants can be loaded efficiently via a base-register relative addressing mode. Since the offset in base-register relative mode is fixed and relatively small (e.g., 16 bits), this limits the maximum size of the constant pool. Thus, in large programs, it is often necessary to use multiple global pointer values in order to be able to address all possible constants. This option causes a warning to be issued whenever this case occurs.

−−warn−once
Only warn once for each undefined symbol, rather than once per module which refers to it.

−−warn−rwx−segments
−−no−warn−rwx−segments
Warn if the linker creates a loadable, non-zero sized segment that has all three of the read, write and execute permission flags set. Such a segment represents a potential security vulnerability. In addition warnings will be generated if a thread local storage segment is created with the execute permission

flag set, regardless of whether or not it has the read and/or write flags set.

These warnings are enabled by default. They can be disabled via the **−−no−warn−rwx−segments** option and re-enabled via the **−−warn−rwx−segments** option.

−−error−rwx−segments
−−no−error−rwx−segments
If the linker is going to generate a warning message about an executable, writeable segment, or an executable TLS segment, then the **−−error−rwx−segments** option will turn this warning into an error instead. The **−−no−error−rwx−segments** option will restore the default behaviour of just generating a warning message.

Note − the **−−error−rwx−segments** option does not by itself turn on warnings about these segments. These warnings are either enabled by default, if the linker was configured that way, or via the **−−warn−rwx−segments** command line option.

−−warn−section−align
Warn if the address of an output section is changed because of alignment. Typically, the alignment will be set by an input section. The address will only be changed if it not explicitly specified; that is, if the `SECTIONS` command does not specify a start address for the section.

−−warn−textrel
Warn if the linker adds DT_TEXTREL to a position-independent executable or shared object.

−−warn−alternate−em
Warn if an object has alternate ELF machine code.

−−warn−unresolved−symbols
If the linker is going to report an unresolved symbol (see the option **−−unresolved−symbols**) it will normally generate an error. This option makes it generate a warning instead.

−−error−unresolved−symbols
This restores the linker's default behaviour of generating errors when it is reporting unresolved symbols.

−−whole−archive
For each archive mentioned on the command line after the **−−whole−archive** option, include every object file in the archive in the link, rather than searching the archive for the required object files. This is normally used to turn an archive file into a shared library, forcing every object to be included in the resulting shared library. This option may be used more than once.

Two notes when using this option from gcc: First, gcc doesn't know about this option, so you have to use **−Wl,−whole−archive**. Second, don't forget to use **−Wl,−no−whole−archive** after your list of archives, because gcc will add its own list of archives to your link and you may not want this flag to affect those as well.

−−wrap=*symbol*
Use a wrapper function for *symbol*. Any undefined reference to *symbol* will be resolved to `__wrap_`*symbol*. Any undefined reference to `__real_`*symbol* will be resolved to *symbol*.

This can be used to provide a wrapper for a system function. The wrapper function should be called `__wrap_`*symbol*. If it wishes to call the system function, it should call `__real_`*symbol*.

Here is a trivial example:

```
void *
__wrap_malloc (size_t c)
{
  printf ("malloc called with %zu\n", c);
  return __real_malloc (c);
}
```

If you link other code with this file using **−−wrap malloc**, then all calls to `malloc` will call the

function `__wrap_malloc` instead. The call to `__real_malloc` in `__wrap_malloc` will call the real `malloc` function.

You may wish to provide a `__real_malloc` function as well, so that links without the **--wrap** option will succeed. If you do this, you should not put the definition of `__real_malloc` in the same file as `__wrap_malloc`; if you do, the assembler may resolve the call before the linker has a chance to wrap it to `malloc`.

Only undefined references are replaced by the linker. So, translation unit internal references to *symbol* are not resolved to `__wrap_`*symbol*. In the next example, the call to `f` in `g` is not resolved to `__wrap_f`.

```
int
f (void)
{
  return 123;
}

int
g (void)
{
  return f();
}
```

--eh-frame-hdr
--no-eh-frame-hdr
Request (**--eh-frame-hdr**) or suppress (**--no-eh-frame-hdr**) the creation of `.eh_frame_hdr` section and ELF `PT_GNU_EH_FRAME` segment header.

--no-ld-generated-unwind-info
Request creation of `.eh_frame` unwind info for linker generated code sections like PLT. This option is on by default if linker generated unwind info is supported. This option also controls the generation of `.sframe` stack trace info for linker generated code sections like PLT.

--enable-new-dtags
--disable-new-dtags
This linker can create the new dynamic tags in ELF. But the older ELF systems may not understand them. If you specify **--enable-new-dtags**, the new dynamic tags will be created as needed and older dynamic tags will be omitted. If you specify **--disable-new-dtags**, no new dynamic tags will be created. By default, the new dynamic tags are not created. Note that those options are only available for ELF systems.

--hash-size=*number*
Set the default size of the linker's hash tables to a prime number close to *number*. Increasing this value can reduce the length of time it takes the linker to perform its tasks, at the expense of increasing the linker's memory requirements. Similarly reducing this value can reduce the memory requirements at the expense of speed.

--hash-style=*style*
Set the type of linker's hash table(s). *style* can be either `sysv` for classic ELF `.hash` section, `gnu` for new style GNU `.gnu.hash` section or `both` for both the classic ELF `.hash` and new style GNU `.gnu.hash` hash tables. The default depends upon how the linker was configured, but for most Linux based systems it will be `both`.

--compress-debug-sections=none
--compress-debug-sections=zlib
--compress-debug-sections=zlib-gnu
--compress-debug-sections=zlib-gabi

−−compress−debug−sections=zstd
On ELF platforms, these options control how DWARF debug sections are compressed using zlib.

−−compress−debug−sections=none doesn't compress DWARF debug sections. **−−compress−debug−sections=zlib−gnu** compresses DWARF debug sections and renames them to begin with **.zdebug** instead of **.debug**. **−−compress−debug−sections=zlib−gabi** also compresses DWARF debug sections, but rather than renaming them it sets the SHF_COMPRESSED flag in the sections' headers.

The **−−compress−debug−sections=zlib** option is an alias for **−−compress−debug−sections=zlib−gabi**.

−−compress−debug−sections=zstd compresses DWARF debug sections using zstd.

Note that this option overrides any compression in input debug sections, so if a binary is linked with **−−compress−debug−sections=none** for example, then any compressed debug sections in input files will be uncompressed before they are copied into the output binary.

The default compression behaviour varies depending upon the target involved and the configure options used to build the toolchain. The default can be determined by examining the output from the linker's **−−help** option.

−−reduce−memory−overheads
This option reduces memory requirements at ld runtime, at the expense of linking speed. This was introduced to select the old O(n^2) algorithm for link map file generation, rather than the new O(n) algorithm which uses about 40% more memory for symbol storage.

Another effect of the switch is to set the default hash table size to 1021, which again saves memory at the cost of lengthening the linker's run time. This is not done however if the **−−hash−size** switch has been used.

The **−−reduce−memory−overheads** switch may be also be used to enable other tradeoffs in future versions of the linker.

−−max−cache−size=*size*
ld normally caches the relocation information and symbol tables of input files in memory with the unlimited size. This option sets the maximum cache size to *size*.

−−build−id
−−build−id=*style*
Request the creation of a `.note.gnu.build-id` ELF note section or a `.buildid` COFF section. The contents of the note are unique bits identifying this linked file. *style* can be `uuid` to use 128 random bits, `sha1` to use a 160−bit SHA1 hash on the normative parts of the output contents, `md5` to use a 128−bit MD5 hash on the normative parts of the output contents, or `0x`*`hexstring`* to use a chosen bit string specified as an even number of hexadecimal digits (`-` and `:` characters between digit pairs are ignored). If *style* is omitted, `sha1` is used.

The `md5` and `sha1` styles produces an identifier that is always the same in an identical output file, but will be unique among all nonidentical output files. It is not intended to be compared as a checksum for the file's contents. A linked file may be changed later by other tools, but the build ID bit string identifying the original linked file does not change.

Passing `none` for *style* disables the setting from any `--build-id` options earlier on the command line.

−−package−metadata=*JSON*
Request the creation of a `.note.package` ELF note section. The contents of the note are in JSON format, as per the package metadata specification. For more information see: https://systemd.io/ELF_PACKAGE_METADATA/ If the JSON argument is missing/empty then this will disable the creation of the metadata note, if one had been enabled by an earlier occurrence of the −−package−metadata option. If the linker has been built with libjansson, then the JSON string will be

validated.

The i386 PE linker supports the **–shared** option, which causes the output to be a dynamically linked library (DLL) instead of a normal executable. You should name the output `*.dll` when you use this option. In addition, the linker fully supports the standard `*.def` files, which may be specified on the linker command line like an object file (in fact, it should precede archives it exports symbols from, to ensure that they get linked in, just like a normal object file).

In addition to the options common to all targets, the i386 PE linker support additional command-line options that are specific to the i386 PE target. Options that take values may be separated from their values by either a space or an equals sign.

––add–stdcall–alias
If given, symbols with a stdcall suffix (@*nn*) will be exported as-is and also with the suffix stripped. [This option is specific to the i386 PE targeted port of the linker]

––base–file *file*
Use *file* as the name of a file in which to save the base addresses of all the relocations needed for generating DLLs with *dlltool*. [This is an i386 PE specific option]

––dll
Create a DLL instead of a regular executable. You may also use **–shared** or specify a `LIBRARY` in a given `.def` file. [This option is specific to the i386 PE targeted port of the linker]

––enable–long–section–names
––disable–long–section–names
The PE variants of the COFF object format add an extension that permits the use of section names longer than eight characters, the normal limit for COFF. By default, these names are only allowed in object files, as fully-linked executable images do not carry the COFF string table required to support the longer names. As a GNU extension, it is possible to allow their use in executable images as well, or to (probably pointlessly!) disallow it in object files, by using these two options. Executable images generated with these long section names are slightly non-standard, carrying as they do a string table, and may generate confusing output when examined with non-GNU PE-aware tools, such as file viewers and dumpers. However, GDB relies on the use of PE long section names to find Dwarf–2 debug information sections in an executable image at runtime, and so if neither option is specified on the command-line, **ld** will enable long section names, overriding the default and technically correct behaviour, when it finds the presence of debug information while linking an executable image and not stripping symbols. [This option is valid for all PE targeted ports of the linker]

––enable–stdcall–fixup
––disable–stdcall–fixup
If the link finds a symbol that it cannot resolve, it will attempt to do "fuzzy linking" by looking for another defined symbol that differs only in the format of the symbol name (cdecl vs stdcall) and will resolve that symbol by linking to the match. For example, the undefined symbol `_foo` might be linked to the function `_foo@12`, or the undefined symbol `_bar@16` might be linked to the function `_bar`. When the linker does this, it prints a warning, since it normally should have failed to link, but sometimes import libraries generated from third-party dlls may need this feature to be usable. If you specify **––enable–stdcall–fixup**, this feature is fully enabled and warnings are not printed. If you specify **––disable–stdcall–fixup**, this feature is disabled and such mismatches are considered to be errors. [This option is specific to the i386 PE targeted port of the linker]

––leading–underscore
––no–leading–underscore
For most targets default symbol-prefix is an underscore and is defined in target's description. By this option it is possible to disable/enable the default underscore symbol-prefix.

––export–all–symbols
If given, all global symbols in the objects used to build a DLL will be exported by the DLL. Note that this is the default if there otherwise wouldn't be any exported symbols. When symbols are explicitly exported via DEF files or implicitly exported via function attributes, the default is to not export

anything else unless this option is given. Note that the symbols `DllMain@12`, `DllEntryPoint@0`, `DllMainCRTStartup@12`, and `impure_ptr` will not be automatically exported. Also, symbols imported from other DLLs will not be re-exported, nor will symbols specifying the DLL's internal layout such as those beginning with `_head_` or ending with `_iname`. In addition, no symbols from `libgcc`, `libstd++`, `libmingw32`, or `crtX.o` will be exported. Symbols whose names begin with `__rtti_` or `__builtin_` will not be exported, to help with C++ DLLs. Finally, there is an extensive list of cygwin-private symbols that are not exported (obviously, this applies on when building DLLs for cygwin targets). These cygwin-excludes are: `_cygwin_dll_entry@12`, `_cygwin_crt0_common@8`, `_cygwin_noncygwin_dll_entry@12`, `_fmode`, `_impure_ptr`, `cygwin_attach_dll`, `cygwin_premain0`, `cygwin_premain1`, `cygwin_premain2`, `cygwin_premain3`, and `environ`. [This option is specific to the i386 PE targeted port of the linker]

−−exclude−symbols *symbol*,*symbol*,...
Specifies a list of symbols which should not be automatically exported. The symbol names may be delimited by commas or colons. [This option is specific to the i386 PE targeted port of the linker]

−−exclude−all−symbols
Specifies no symbols should be automatically exported. [This option is specific to the i386 PE targeted port of the linker]

−−file−alignment
Specify the file alignment. Sections in the file will always begin at file offsets which are multiples of this number. This defaults to 512. [This option is specific to the i386 PE targeted port of the linker]

−−heap *reserve*
−−heap *reserve*,*commit*
Specify the number of bytes of memory to reserve (and optionally commit) to be used as heap for this program. The default is 1MB reserved, 4K committed. [This option is specific to the i386 PE targeted port of the linker]

−−image−base *value*
Use *value* as the base address of your program or dll. This is the lowest memory location that will be used when your program or dll is loaded. To reduce the need to relocate and improve performance of your dlls, each should have a unique base address and not overlap any other dlls. The default is 0x400000 for executables, and 0x10000000 for dlls. [This option is specific to the i386 PE targeted port of the linker]

−−kill−at
If given, the stdcall suffixes (@*nn*) will be stripped from symbols before they are exported. [This option is specific to the i386 PE targeted port of the linker]

−−large−address−aware
If given, the appropriate bit in the "Characteristics" field of the COFF header is set to indicate that this executable supports virtual addresses greater than 2 gigabytes. This should be used in conjunction with the /3GB or /USERVA=*value* megabytes switch in the "[operating systems]" section of the BOOT.INI. Otherwise, this bit has no effect. [This option is specific to PE targeted ports of the linker]

−−disable−large−address−aware
Reverts the effect of a previous **−−large−address−aware** option. This is useful if **−−large−address−aware** is always set by the compiler driver (e.g. Cygwin gcc) and the executable does not support virtual addresses greater than 2 gigabytes. [This option is specific to PE targeted ports of the linker]

−−major−image−version *value*
Sets the major number of the "image version". Defaults to 1. [This option is specific to the i386 PE targeted port of the linker]

−−major−os−version *value*
Sets the major number of the "os version". Defaults to 4. [This option is specific to the i386 PE targeted port of the linker]

−−major−subsystem−version *value*
Sets the major number of the "subsystem version". Defaults to 4. [This option is specific to the i386 PE targeted port of the linker]

−−minor−image−version *value*
Sets the minor number of the "image version". Defaults to 0. [This option is specific to the i386 PE targeted port of the linker]

−−minor−os−version *value*
Sets the minor number of the "os version". Defaults to 0. [This option is specific to the i386 PE targeted port of the linker]

−−minor−subsystem−version *value*
Sets the minor number of the "subsystem version". Defaults to 0. [This option is specific to the i386 PE targeted port of the linker]

−−output−def *file*
The linker will create the file *file* which will contain a DEF file corresponding to the DLL the linker is generating. This DEF file (which should be called `*.def`) may be used to create an import library with `dlltool` or may be used as a reference to automatically or implicitly exported symbols. [This option is specific to the i386 PE targeted port of the linker]

−−enable−auto−image−base
−−enable−auto−image−base=*value*
Automatically choose the image base for DLLs, optionally starting with base *value*, unless one is specified using the `−−image−base` argument. By using a hash generated from the dllname to create unique image bases for each DLL, in-memory collisions and relocations which can delay program execution are avoided. [This option is specific to the i386 PE targeted port of the linker]

−−disable−auto−image−base
Do not automatically generate a unique image base. If there is no user-specified image base (`−−image−base`) then use the platform default. [This option is specific to the i386 PE targeted port of the linker]

−−dll−search−prefix *string*
When linking dynamically to a dll without an import library, search for `<string><basename>.dll` in preference to `lib<basename>.dll`. This behaviour allows easy distinction between DLLs built for the various "subplatforms": native, cygwin, uwin, pw, etc. For instance, cygwin DLLs typically use `−−dll−search−prefix=cyg`. [This option is specific to the i386 PE targeted port of the linker]

−−enable−auto−import
Do sophisticated linking of `_symbol` to `__imp__symbol` for DATA imports from DLLs, thus making it possible to bypass the dllimport mechanism on the user side and to reference unmangled symbol names. [This option is specific to the i386 PE targeted port of the linker]

The following remarks pertain to the original implementation of the feature and are obsolete nowadays for Cygwin and MinGW targets.

Note: Use of the 'auto−import' extension will cause the text section of the image file to be made writable. This does not conform to the PE-COFF format specification published by Microsoft.

Note − use of the 'auto−import' extension will also cause read only data which would normally be placed into the .rdata section to be placed into the .data section instead. This is in order to work around a problem with consts that is described here: http://www.cygwin.com/ml/cygwin/2004−09/msg01101.html

Using 'auto−import' generally will 'just work' −− but sometimes you may see this message:

"variable '<var>' can't be auto-imported. Please read the documentation for ld's `--enable-auto-import` for details."

This message occurs when some (sub)expression accesses an address ultimately given by the sum of two constants (Win32 import tables only allow one). Instances where this may occur include accesses to member fields of struct variables imported from a DLL, as well as using a constant index into an array variable imported from a DLL. Any multiword variable (arrays, structs, long long, etc) may trigger this error condition. However, regardless of the exact data type of the offending exported variable, ld will always detect it, issue the warning, and exit.

There are several ways to address this difficulty, regardless of the data type of the exported variable:

One way is to use --enable-runtime-pseudo-reloc switch. This leaves the task of adjusting references in your client code for runtime environment, so this method works only when runtime environment supports this feature.

A second solution is to force one of the 'constants' to be a variable -- that is, unknown and un-optimizable at compile time. For arrays, there are two possibilities: a) make the indexee (the array's address) a variable, or b) make the 'constant' index a variable. Thus:

```
extern type extern_array[];
extern_array[1] -->
   { volatile type *t=extern_array; t[1] }
```

or

```
extern type extern_array[];
extern_array[1] -->
   { volatile int t=1; extern_array[t] }
```

For structs (and most other multiword data types) the only option is to make the struct itself (or the long long, or the ...) variable:

```
extern struct s extern_struct;
extern_struct.field -->
   { volatile struct s *t=&extern_struct; t->field }
```

or

```
extern long long extern_ll;
extern_ll -->
  { volatile long long * local_ll=&extern_ll; *local_ll }
```

A third method of dealing with this difficulty is to abandon 'auto-import' for the offending symbol and mark it with `__declspec(dllimport)`. However, in practice that requires using compile-time #defines to indicate whether you are building a DLL, building client code that will link to the DLL, or merely building/linking to a static library. In making the choice between the various methods of resolving the 'direct address with constant offset' problem, you should consider typical real-world usage:

Original:

```
--foo.h
extern int arr[];
--foo.c
#include "foo.h"
void main(int argc, char **argv){
  printf("%d\n",arr[1]);
}
```

Solution 1:

```
--foo.h
extern int arr[];
--foo.c
#include "foo.h"
void main(int argc, char **argv){
  /* This workaround is for win32 and cygwin; do not "optimize" */
  volatile int *parr = arr;
  printf("%d\n",parr[1]);
}
```

Solution 2:

```
--foo.h
/* Note: auto-export is assumed (no __declspec(dllexport)) */
#if (defined(_WIN32) || defined(__CYGWIN__)) && \
  !(defined(FOO_BUILD_DLL) || defined(FOO_STATIC))
#define FOO_IMPORT __declspec(dllimport)
#else
#define FOO_IMPORT
#endif
extern FOO_IMPORT int arr[];
--foo.c
#include "foo.h"
void main(int argc, char **argv){
  printf("%d\n",arr[1]);
}
```

A fourth way to avoid this problem is to re-code your library to use a functional interface rather than a data interface for the offending variables (e.g. **set_foo()** and **get_foo()** accessor functions).

−−disable−auto−import
Do not attempt to do sophisticated linking of `_symbol` to `__imp__symbol` for DATA imports from DLLs. [This option is specific to the i386 PE targeted port of the linker]

−−enable−runtime−pseudo−reloc
If your code contains expressions described in −−enable−auto−import section, that is, DATA imports from DLL with non-zero offset, this switch will create a vector of 'runtime pseudo relocations' which can be used by runtime environment to adjust references to such data in your client code. [This option is specific to the i386 PE targeted port of the linker]

−−disable−runtime−pseudo−reloc
Do not create pseudo relocations for non-zero offset DATA imports from DLLs. [This option is specific to the i386 PE targeted port of the linker]

−−enable−extra−pe−debug
Show additional debug info related to auto-import symbol thunking. [This option is specific to the i386 PE targeted port of the linker]

−−section−alignment
Sets the section alignment. Sections in memory will always begin at addresses which are a multiple of this number. Defaults to 0x1000. [This option is specific to the i386 PE targeted port of the linker]

−−stack *reserve*
−−stack *reserve*,*commit*
Specify the number of bytes of memory to reserve (and optionally commit) to be used as stack for this program. The default is 2MB reserved, 4K committed. [This option is specific to the i386 PE targeted port of the linker]

−−subsystem *which*
−−subsystem *which*:*major*
−−subsystem *which*:*major*.*minor*
Specifies the subsystem under which your program will execute. The legal values for *which* are `native`, `windows`, `console`, `posix`, and `xbox`. You may optionally set the subsystem version also. Numeric values are also accepted for *which*. [This option is specific to the i386 PE targeted port of the linker]

The following options set flags in the `DllCharacteristics` field of the PE file header: [These options are specific to PE targeted ports of the linker]

−−high−entropy−va
−−disable−high−entropy−va
Image is compatible with 64−bit address space layout randomization (ASLR). This option is enabled by default for 64−bit PE images.

This option also implies **−−dynamicbase** and **−−enable−reloc−section**.

−−dynamicbase
−−disable−dynamicbase
The image base address may be relocated using address space layout randomization (ASLR). This feature was introduced with MS Windows Vista for i386 PE targets. This option is enabled by default but can be disabled via the **−−disable−dynamicbase** option. This option also implies **−−enable−reloc−section**.

−−forceinteg
−−disable−forceinteg
Code integrity checks are enforced. This option is disabled by default.

−−nxcompat
−−disable−nxcompat
The image is compatible with the Data Execution Prevention. This feature was introduced with MS Windows XP SP2 for i386 PE targets. The option is enabled by default.

−−no−isolation
−−disable−no−isolation
Although the image understands isolation, do not isolate the image. This option is disabled by default.

−−no−seh
−−disable−no−seh
The image does not use SEH. No SE handler may be called from this image. This option is disabled by default.

−−no−bind
−−disable−no−bind
Do not bind this image. This option is disabled by default.

−−wdmdriver
−−disable−wdmdriver
The driver uses the MS Windows Driver Model. This option is disabled by default.

−−tsaware
−−disable−tsaware
The image is Terminal Server aware. This option is disabled by default.

−−insert−timestamp
−−no−insert−timestamp
Insert a real timestamp into the image. This is the default behaviour as it matches legacy code and it means that the image will work with other, proprietary tools. The problem with this default is that it will result in slightly different images being produced each time the same sources are linked. The option **−−no−insert−timestamp** can be used to insert a zero value for the timestamp, this ensuring

that binaries produced from identical sources will compare identically.

If **−−insert−timestamp** is active then the time inserted is either the time that the linking takes place or, if the `SOURCE_DATE_EPOCH` environment variable is defined, the number of seconds since Unix epoch as specified by that variable.

−−enable−reloc−section
−−disable−reloc−section
Create the base relocation table, which is necessary if the image is loaded at a different image base than specified in the PE header. This option is enabled by default.

The C6X uClinux target uses a binary format called DSBT to support shared libraries. Each shared library in the system needs to have a unique index; all executables use an index of 0.

−−dsbt−size *size*
This option sets the number of entries in the DSBT of the current executable or shared library to *size*. The default is to create a table with 64 entries.

−−dsbt−index *index*
This option sets the DSBT index of the current executable or shared library to *index*. The default is 0, which is appropriate for generating executables. If a shared library is generated with a DSBT index of 0, the `R_C6000_DSBT_INDEX` relocs are copied into the output file.

The **−−no−merge−exidx−entries** switch disables the merging of adjacent exidx entries in frame unwind info.

−−branch−stub
This option enables linker branch relaxation by inserting branch stub sections when needed to extend the range of branches. This option is usually not required since C−SKY supports branch and call instructions that can access the full memory range and branch relaxation is normally handled by the compiler or assembler.

−−stub−group−size=*N*
This option allows finer control of linker branch stub creation. It sets the maximum size of a group of input sections that can be handled by one stub section. A negative value of *N* locates stub sections after their branches, while a positive value allows stub sections to appear either before or after the branches. Values of **1** or **−1** indicate that the linker should choose suitable defaults.

The 68HC11 and 68HC12 linkers support specific options to control the memory bank switching mapping and trampoline code generation.

−−no−trampoline
This option disables the generation of trampoline. By default a trampoline is generated for each far function which is called using a `jsr` instruction (this happens when a pointer to a far function is taken).

−−bank−window *name*
This option indicates to the linker the name of the memory region in the **MEMORY** specification that describes the memory bank window. The definition of such region is then used by the linker to compute paging and addresses within the memory window.

The following options are supported to control handling of GOT generation when linking for 68K targets.

−−got=*type*
This option tells the linker which GOT generation scheme to use. *type* should be one of **single**, **negative**, **multigot** or **target**. For more information refer to the Info entry for *ld*.

The following options are supported to control microMIPS instruction generation and branch relocation checks for ISA mode transitions when linking for MIPS targets.

−−insn32

−−no−insn32
These options control the choice of microMIPS instructions used in code generated by the linker, such as that in the PLT or lazy binding stubs, or in relaxation. If **−−insn32** is used, then the linker only uses 32−bit instruction encodings. By default or if **−−no−insn32** is used, all instruction encodings are used, including 16−bit ones where possible.

−−ignore−branch−isa
−−no−ignore−branch−isa
These options control branch relocation checks for invalid ISA mode transitions. If **−−ignore−branch−isa** is used, then the linker accepts any branch relocations and any ISA mode transition required is lost in relocation calculation, except for some cases of `BAL` instructions which meet relaxation conditions and are converted to equivalent `JALX` instructions as the associated relocation is calculated. By default or if **−−no−ignore−branch−isa** is used a check is made causing the loss of an ISA mode transition to produce an error.

−−compact−branches
−−no−compact−branches
These options control the generation of compact instructions by the linker in the PLT entries for MIPS R6.

For the pdp11−aout target, three variants of the output format can be produced as selected by the following options. The default variant for pdp11−aout is the **−−omagic** option, whereas for other targets **−−nmagic** is the default. The **−−imagic** option is defined only for the pdp11−aout target, while the others are described here as they apply to the pdp11−aout target.

−N
−−omagic
Mark the output as `OMAGIC` (0407) in the *a.out* header to indicate that the text segment is not to be write-protected and shared. Since the text and data sections are both readable and writable, the data section is allocated immediately contiguous after the text segment. This is the oldest format for PDP11 executable programs and is the default for **ld** on PDP11 Unix systems from the beginning through 2.11BSD.

−n
−−nmagic
Mark the output as `NMAGIC` (0410) in the *a.out* header to indicate that when the output file is executed, the text portion will be read-only and shareable among all processes executing the same file. This involves moving the data areas up to the first possible 8K byte page boundary following the end of the text. This option creates a *pure executable* format.

−z
−−imagic
Mark the output as `IMAGIC` (0411) in the *a.out* header to indicate that when the output file is executed, the program text and data areas will be loaded into separate address spaces using the split instruction and data space feature of the memory management unit in larger models of the PDP11. This doubles the address space available to the program. The text segment is again pure, write-protected, and shareable. The only difference in the output format between this option and the others, besides the magic number, is that both the text and data sections start at location 0. The **−z** option selected this format in 2.11BSD. This option creates a *separate executable* format.

−−no−omagic
Equivalent to **−−nmagic** for pdp11−aout.

ENVIRONMENT

You can change the behaviour of **ld** with the environment variables `GNUTARGET`, `LDEMULATION` and `COLLECT_NO_DEMANGLE`.

`GNUTARGET` determines the input-file object format if you don't use **−b** (or its synonym **−−format**). Its value should be one of the BFD names for an input format. If there is no `GNUTARGET` in the environment, **ld** uses the natural format of the target. If `GNUTARGET` is set to `default` then BFD attempts to discover

the input format by examining binary input files; this method often succeeds, but there are potential ambiguities, since there is no method of ensuring that the magic number used to specify object-file formats is unique. However, the configuration procedure for BFD on each system places the conventional format for that system first in the search-list, so ambiguities are resolved in favor of convention.

`LDEMULATION` determines the default emulation if you don't use the **−m** option. The emulation can affect various aspects of linker behaviour, particularly the default linker script. You can list the available emulations with the **−−verbose** or **−V** options. If the **−m** option is not used, and the `LDEMULATION` environment variable is not defined, the default emulation depends upon how the linker was configured.

Normally, the linker will default to demangling symbols. However, if `COLLECT_NO_DEMANGLE` is set in the environment, then it will default to not demangling symbols. This environment variable is used in a similar fashion by the `gcc` linker wrapper program. The default may be overridden by the **−−demangle** and **−−no−demangle** options.

SEE ALSO

ar (1), **nm** (1), **objcopy** (1), **objdump** (1), **readelf** (1) and the Info entries for *binutils* and *ld*.

COPYRIGHT