

NAME

ld – Using 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*, or the manual *ld: the GNU linker*, 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. Note that specifying a script in this way merely augments the main linker script; use the **-T** option to replace the default linker script entirely.

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,--startgroup foo.o bar.o -Wl,--endgroup
```

This is important, because otherwise the compiler driver program may silently drop the linker options, resulting in a bad link.

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

-akeyword

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.

-Aarchitecture

--architecture=architecture

In the current release of **ld**, this option is useful only for the Intel 960 family of architectures. In that **ld** configuration, the *architecture* argument identifies the particular architecture in the 960 family, enabling some safeguards and modifying the archive-library search path.

Future releases of **ld** may support similar functionality for other architecture families.

-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.

-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).

-E
--export-dynamic

When creating a dynamically linked executable, 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 this option, 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 version script to control what symbols should be added to the dynamic symbol table if the output format supports it. See the description of **--version-script** in `@ref{VERSION}`.

-EB

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

-EL

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

-f
--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 ECOFF which supports 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 the 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 *archive*

--library=*archive*

Add archive file *archive* to the list of files to link. This option may be used any number of times. **ld** will search its path-list for occurrences of `libarchive.a` for every *archive* specified.

On systems which support shared libraries, **ld** may also search for libraries with extensions other than `.a`. Specifically, on ELF and SunOS systems, **ld** will search a directory for a library with an extension of `.so` before searching for one with an extension of `.a`. By convention, a `.so` extension indicates a shared library.

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.

If *searchdir* begins with =, then the = will be replaced by the *sysroot prefix*, a path 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.

-memulation

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.

-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 and symbols 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.

-n

--nmagic

Turn off page alignment of sections, and mark the output as `NMAGIC` if possible.

-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`.

--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.

-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.

-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.

-r

—relocateable

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.

-t

—trace

Print the names of the input files as **ld** processes them.

-T *scriptfile*

—script=*scriptfile*

Use *scriptfile* as the linker script. This script replaces **ld**'s default linker script (rather than adding to it), 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.

-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.

-Ur

For anything other than C++ programs, this option is equivalent to **-r**: it generates relocatable output—i.e., an output file that can in turn serve as input to **ld**. When linking C++ programs, **-Ur** *does* resolve references to constructors, unlike **-r**. 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.

—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.

-x

--discard-all

Delete all local symbols.

-X

--discard-locals

Delete all temporary local symbols. For most targets, this is all local symbols whose names begin with **L**.

-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 *initfirst*, *interpose*, *loadfltr*, *nodefaultlib*, *nodelete*, *nodlopen*, *nodump*, *now*, *origin*, *combrelloc*, *nocombrelloc* and *nocopyreloc*. The other keywords are ignored for Solaris compatibility. *initfirst* marks the object to be initialized first at runtime before any other objects. *interpose* marks the object that its symbol table interposes before all symbols but the primary executable. *loadfltr* marks the object that its filters be processed immediately at runtime. *nodefaultlib* marks the object that the search for dependencies of this object will ignore any default library search paths. *nodelete* marks the object shouldn't be unloaded at runtime. *nodlopen* marks the object not available to *dlopen*. *nodump* marks the object can not be dumped by *dldump*. *now* marks the object with the non-lazy runtime binding. *origin* marks the object may contain *\$ORIGIN*. *defs* disallows undefined symbols. *muldefs* allows multiple definitions. *combrelloc* combines multiple reloc sections and sorts them to make dynamic symbol lookup caching possible. *nocombrelloc* disables multiple reloc sections combining. *nocopyreloc* disables production of copy relocations.

-(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 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.

-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 run-time linker to handle lookups in this object and its dependencies to be performed only inside the group. **--no-undefined** 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.

-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.

--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**.

--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. The remaining files contain references to the symbol.

--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.

--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*.

--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.

--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.

--embedded-relocs

This option is only meaningful when linking MIPS embedded PIC code, generated by the **--membedded-pic** option to the GNU compiler and assembler. It causes the linker to create a table which may be used at runtime to relocate any data which was statically initialized to pointer values. See the code in `testsuite/ld-empic` for details.

--fatal-warnings

Treat all warnings as errors.

--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.

--no-gc-sections

--gc-sections

Enable garbage collection of unused input sections. It is ignored on targets that do not support this option. This option is not compatible with **-r**, nor should it be used with dynamic linking. The default behaviour (of not performing this garbage collection) can be restored by specifying **--no-gc-sections** on the command line.

--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.

--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

Normally when creating a non-symbolic shared library, undefined symbols are allowed and left to be resolved by the runtime loader. This option disallows such undefined symbols if they come from regular object files. The switch **--no-allow-shlib-undefined** controls the behaviour for shared objects being linked into the shared library.

--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

Allow (the default) or disallow undefined symbols in shared objects. The setting of this switch overrides **--no-undefined** where shared objects are concerned. Thus if **--no-undefined** is set but **--no-allow-shlib-undefined** is not, the net result will be that undefined symbols in regular object files will trigger an error, but undefined symbols in shared objects will be ignored.

The reason that **--allow-shlib-undefined** is the default is that the shared object being specified at link time may not be the same one that is available at load time, so the symbols might actually be resolvable at load time. Plus there are some systems, (eg BeOS) where undefined symbols in shared libraries is normal since the kernel patches them at load time to select which function is most appropriate for the current architecture. eg. to dynamically select an appropriate memset function. Apparently it is also normal for HPPA shared libraries to have undefined symbols.

--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.

--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-whole-archive

Turn off the effect of the **--whole-archive** option for subsequent archive files.

--noinhbit-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.

--format *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 **--format** 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.

-qmagic

This option is ignored for Linux compatibility.

-Qy

This option is ignored for SVR4 compatibility.

--relax

An option with machine dependent effects. This option is only supported on a few targets.

On some platforms, the **--relax** option performs global optimizations that become possible when the linker resolves addressing in the program, such as relaxing address modes and synthesizing new instructions in the output object file.

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 this is not supported, **--relax** is 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. 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 filesystems.

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.

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. It is for the native linker only.

3. On an ELF system, if the **-rpath** and **rpath-link** options were not used, search the contents of the environment variable **LD_RUN_PATH**. It is for the native linker only.
4. On SunOS, if the **-rpath** option was not used, search any directories specified using **-L** options.
5. For a native linker, 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. The default directories, normally **/lib** and **/usr/lib**.
8. For a native linker on an ELF system, if the file **/etc/ld.so.conf** exists, the list of directories found in that file.

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

This option tells **ld** to sort the common symbols by size when it places them in the appropriate output sections. First come all the one byte symbols, then all the two byte, then all the four byte, and then everything else. This is to prevent gaps between symbols due to alignment constraints.

--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 create 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.

--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*.

---dll-verbose

---verbose

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.

---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 meaningful on ELF platforms which support shared libraries.

---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-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-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.

--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 (int c)
{
    printf ("malloc called with %ld\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`.

--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 dynamic tags will be created as needed. 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.

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-stdcall-fixup

--disable-stdcall-fixup

If the linker 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]

--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-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. Symbols explicitly listed in a `.def` file are still exported, regardless of this option. [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 amount 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]
- 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]

--out-implib *file*

The linker will create the file *file* which will contain an import lib corresponding to the DLL the linker is generating. This import lib (which should be called *.dll.a or *.a) may be used to link clients against the generated DLL; this behavior makes it possible to skip a separate `dlltool` import library creation step. [This option is specific to the i386 PE targeted port of the linker]

--enable-auto-image-base

Automatically choose the image base for DLLs, 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><base-name>.dll` in preference to `lib<basename>.dll`. This behavior 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, and create the necessary thunking symbols when building the import libraries with those DATA exports. This 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). [This option is specific to the i386 PE targeted port of the linker]

--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 is the default. [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 thinking. [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 amount 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, and posix. You may optionally set the subsystem version also. [This option is specific to the i386 PE targeted port of the linker]

ENVIRONMENT

You can change the behavior 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

Copyright (c) 1991, 92, 93, 94, 95, 96, 97, 98, 99, 2000, 2001, 2002, 2003 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".