NAME

ld – link editor for object files

SYNOPSIS

/usr/ccs/bin/ld [**–a** | **–r**] [**–b**] [**–G**] [**–i**] [**–m**] [**–s**] [**–t**] [**–V**]
[**–B dynamic** | **static**] [**–B local**] [**–B reduce**] [**–B symbolic**] [**–d y** | **n**]
[**–D** *token*] [**–e** *epsym*] [**–F** *name*] [**–f** *name*] [**–h** *name*] [**–I** *name*]
[**–L** *path*] [**–l** *x*] [**–M** *mapfile*] [**–o** *outfile*] [**–Q y** | **n**] [**–R** *path*]
[**–u** *symname*] [**–Y** P,*dirlist*] [**–z defs** | **nodefs**] [**–z muldefs**] [**–z noversion**]
[**–z text**] *filename . . .*

AVAILABILITY

SUNWtoo

DESCRIPTION

The **ld** command combines relocatable object files, performs relocation, and resolves external symbols. **ld** operates in two modes, static or dynamic, as governed by the **–d** option. In static mode, **–dn**, relocatable object files given as arguments are combined to produce an executable object file; if the **–r** option is specified, relocatable object files are combined to produce one relocatable object file. In dynamic mode, **–dy**, the default, relocatable object files given as arguments are combined to produce an executable object file that will be linked at execution with any shared object files given as arguments; if the **–G** option is specified, relocatable object files are combined to produce a shared object. In all cases, the output of **ld** is left in **a.out** by default.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. The library may be either a relocatable archive or a shared object. For an archive library, only those routines defining an unresolved external reference are loaded. The archive library symbol table (see **ar**(4)) is searched sequentially with as many passes as are necessary to resolve external references that can be satisfied by library members. Thus, the ordering of members in the library is functionally unimportant, unless there exist multiple library members defining the same external symbol. A shared object consists of a single entity all of whose references must be resolved within the executable being built or within other shared objects with which it is linked.

OPTIONS

–a In static mode only, produce an executable object file; give errors for undefined references. This is the default behavior for static mode. **–a** may not be used with the **–r** option.

–r Combine relocatable object files to produce one relocatable object file. **ld** will not complain about unresolved references. This option cannot be used in dynamic mode or with **–a**.

–b In dynamic mode only, when creating an executable, do not do special processing for relocations that reference symbols in shared objects. Without the **–b** option, the link editor creates special position-independent relocations for references to functions defined in shared objects and arranges for data objects defined in shared objects to be copied into the memory image of the executable by the runtime linker. With the **–b** option, the output code may be more efficient, but it will be less sharable.

−G In dynamic mode only, produce a shared object. Undefined symbols are allowed.

−i Ignore **LD_LIBRARY_PATH** setting. This option is useful when an **LD_LIBRARY_PATH** setting is in effect to influence the runtime library search, which would interfere with the link editing being performed.

−m Produce a memory map or listing of the input/output sections, together with any non-fatal multiply defined symbols, on the standard output.

−s Strip symbolic information from the output file. Any debugging information, that is *.debug*, *.line*, and *.stab* sections, and their associated relocation entries will be removed. Except for relocatable files or shared objects, the symbol table and string table sections will also be removed from the output object file.

−t Turn off the warning about multiply defined symbols that are not the same size.

−V Output a message giving information about the version of **ld** being used.

−B dynamic | static
Options governing library inclusion. −**B dynamic** is valid in dynamic mode only. These options may be specified any number of times on the command line as toggles: if the −**B static** option is given, no shared objects will be accepted until −**B dynamic** is seen. See also the −**l** option.

−B local Cause any global symbols, not assigned to a version definition, to be reduced to local. Version definitions can be supplied via a *mapfile*, and indicate the global symbols that should remain visible in the generated object. This option achieves the same symbol reduction as the *auto-reduction* directive available as part of a *mapfile* version definition, and may be useful when combining versioned and non-versioned relocatable objects.

−B reduce When generating a relocatable object, cause the reduction of symbolic information as defined by any version definitions. Version definitions can be supplied via a *mapfile*, and indicate the global symbols that should remain visible in the generated object. By default, when generating a relocatable object, version definitions are only recorded in the output image. The actual reduction of symbolic information will be carried out when the object itself is used in the construction of a dynamic executable or shared object. When creating a dynamic executable or shared object, this option is applied automatically.

−B symbolic In dynamic mode only, when building a shared object, bind references to global symbols to their definitions within the object, if definitions are available. Normally, references to global symbols within shared objects are not bound until runtime, even if definitions are available, so that definitions of the same symbol in an executable or other shared objects can override the object's own definition. **ld** will issue warnings for undefined symbols unless −**z defs** overrides.

–D *token,token,. . .*
Print debugging information, as specified by each *token*, to the standard error. The special token *help* indicates the full list of tokens available.

–e *epsym* Set the entry point address for the output file to be that of the symbol *epsym*.

–F *name* Useful only when building a shared object. Specifies that the symbol table of the shared object is used as a "filter" on the symbol table of the shared object specified by *name*.

–f *name* Useful only when building a shared object. Specifies that the symbol table of the shared object is used as an “auxiliary filter” on the symbol table of the shared object specified by *name*.

–h *name* In dynamic mode only, when building a shared object, record *name* in the object’s dynamic section. *name* will be recorded in executables that are linked with this object rather than the object’s UNIX System file name. Accordingly, *name* will be used by the runtime linker as the name of the shared object to search for at runtime.

–I *name* When building an executable, use *name* as the path name of the interpreter to be written into the program header. The default in static mode is no interpreter; in dynamic mode, the default is the name of the runtime linker, **/usr/lib/ld.so.1**. Either case may be overridden by –**I***name*. **exec** will load this interpreter when it loads the **a.out** and will pass control to the interpreter rather than to the **a.out** directly.

–L *path* Add *path* to the library search directories. **ld** searches for libraries first in any directories specified by the –**L** options, and then in the standard directories. This option is useful only if it precedes the –**l** options to which it applies on the command line. The environment variable **LD_LIBRARY_PATH** may be used to supplement the library search path (see **LD_LIBRARY_PATH** below).

–l *x* Search a library **lib***x***.so** or **lib***x***.a**, the conventional names for shared object and archive libraries, respectively. In dynamic mode, unless the –**B static** option is in effect, **ld** searches each directory specified in the library search path for a file **lib***x***.so** or **lib***x***.a**. The directory search stops at the first directory containing either. **ld** chooses the file ending in **.so** if –**l***x* expands to two files whose names are of the form **lib***x***.so** and **lib***x***.a**. If no **lib***x***.so** is found, then **ld** accepts **lib***x***.a**. In static mode, or when the –**B static** option is in effect, **ld** selects only the file ending in **.a**. A library is searched when its name is encountered, so the placement of –**l** is significant.

–M *mapfile* Read *mapfile* as a text file of directives to **ld**. This option may be specified multiple times. If *mapfile* is a directory then all regular files (as defined by **stat**(2)) within the directory will be processed. See *Linker and Libraries Guide* for description of mapfiles.

–o *outfile* Produce an output object file named *outfile*. The name of the default object file is **a.out**.

–Q y|n Under –**Qy**, an **ident** string is added to the *.comment* section of the output file to identify the version of the link editor used to create the file. This will result in multiple **ld idents** when there have been multiple linking steps, such as when using **ld** –**r**. This is identical with the default action of the **cc** command. –**Qn** suppresses version identification.

–**R** *path* A colon-separated list of directories used to specify library search directories to the runtime linker. If present and not null, it is recorded in the output object file and passed to the runtime linker. Multiple instances of this option are concatenated together with each *path* separated by a colon.

–**u** *symname* Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from an archive library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the command line is significant; it must be placed before the library that will define the symbol.

–**Y P,***dirlist* Change the default directories used for finding libraries. *dirlist* is a colon-separated path list.

–z defs Force a fatal error if any undefined symbols remain at the end of the link. This is the default when building an executable. It is also useful when building a shared object to assure that the object is self-contained, that is, that all its symbolic references are resolved internally.

–z muldefs Allows multiple symbol definitions. By default, multiple symbol definitions occurring between relocatable objects will result in a fatal error condition. This option suppresses the error condition, and allows the first symbol definition to be taken.

–z nodefs Allow undefined symbols. This is the default when building a shared object. When used with executables, the behavior of references to such "undefined symbols" is unspecified.

–z noversion
Do not record any versioning sections. Any version sections or associated *.dynamic* section entries will not be generated in the output image.

–z text In dynamic mode only, force a fatal error if any relocations against non-writable, allocatable sections remain.

ENVIRONMENT

LD_LIBRARY_PATH
A list of directories in which to search for libraries specified with the –**l** option. Multiple directories are separated by a colon. In the most general case, it will contain two directory lists separated by a semicolon:
*dirlist1***;***dirlist2*

If **ld** is called with any number of occurrences of –**L**, as in:

ld . . . –**L***path1* . . . –**L***pathn* . . .

then the search path ordering is:

dirlist1 path1 . . . *pathn dirlist2* **LIBPATH**

When the list of directories does not contain a semicolon, it is interpreted as *dirlist2*.

LD_LIBRARY_PATH is also used to specify library search directories to the runtime linker. That is, if **LD_LIBRARY_PATH** exists in the environment, the runtime linker will search the directories named in it, before its default directory, for shared objects to be linked with the program at execution.

Note: When running a set-user-ID or set-group-ID program, the runtime linker will only search for libraries in any full pathname specified within the executable as a result of a runpath being specified when the executable was constructed, or in **/usr/lib**. Any library dependencies specified as relative pathnames will be silently ignored.

LD_OPTIONS

A default set of options to **ld**. **LD_OPTIONS** is interpreted by **ld** just as though its value had been placed on the command line, immediately following the name used to invoke **ld**, as in:

ld $LD_OPTIONS . . . *other-arguments* . . .

LD_PRELOAD

A list of shared objects that are to be interpreted by the runtime linker. The specified shared objects are linked in after the program being executed and before any other shared objects that the program references.

Note: When running a set-user-ID or set-group-ID program, this option has some restrictions. The runtime linker will only search for these shared objects in any full pathname specified within the executable as a result of a runpath being specified when the executable was constructed, or in **/usr/lib**. Any shared object specified as a relative, or full pathname, will be silently ignored.

LD_RUN_PATH

An alternative mechanism for specifying a runpath to the link editor (see **-R** option). If both **LD_RUN_PATH** and the **-R** option are specified, **-R** supersedes.

LD_DEBUG

Provide a list of tokens that will cause the runtime linker to print debugging information to the standard error. The special token *help* indicates the full list of tokens available. The environment variable **LD_DEBUG_OUTPUT** may also be supplied to specify a file to which the debugging information is sent. The filename will be suffixed with the process id of the application generating the debugging information.

LD_PROFILE

A shared object that will be profiled by the runtime linker. When profiling is enabled, a profiling buffer file is created and mapped. The name of the buffer file is the name of the shared object being profiled with a `**.profile**´ extension. By default this buffer is placed under **/var/tmp**. The environment variable **LD_PROFILE_OUTPUT** may also be supplied to indicate an alternative directory in which to place the profiling buffer. This buffer contains **profil**(2) and call count information similar to the *gmon.out* information generated by programs that have been linked with the **-xpg** option of **cc**. Any applications that use the named shared object and run while this environment variable is set, will accumulate data in the profile buffer. The profile buffer information may be examined using **gprof**(1).

Note that environment variable-names beginning with the characters `**LD_**´ are reserved for possible future enhancements to **ld**.

FILES

lib*x***.so**	libraries
lib*x***.a**	libraries
a.out	output file
LIBPATH	usually **/usr/ccs/lib:/usr/lib**

SEE ALSO

as(1), **cc**(1B), **gprof**(1), **ld**(1B), **pvs**(1), **exec**(2), **exit**(2), **profil**(2), **elf**(3E), **end**(3C), **exit**(3C), **a.out**(4), **ar**(4)

Linker and Libraries Guide
Binary Compatibility Guide

NOTES

Options No Longer Supported

The following SunOS 4.*x.y* options do not have any replacement in this release: −**B nosymbolic** (this is now the default if −**B symbolic** is not used), −**d**, −**dc**, and −**dp**, (these are now the default, see −**b** above to override the default), −**M**, −**S**, −**t**, −**x**, −**X**, and −**y***sym*.

The following SunOS 4.*x.y* options are not supported: −**align** *datum*, −**A** *name*, −**D** , −**p**, −**T**[**text**] *hex*, −**T data***hex*. Much of the functionality of these options can be achieved using the −**M***mapfile* option.

Obsolete Options

The following SunOS 4.*x.y* options are obsolete in this release: −**n**, −**N**, and −**z**.