
## *Rocky Enterprise Linux 9.2 Manual Pages on command 'dpkg-shlibdeps.1'*

***C:\>man dpkg-shlibdeps.1***

dpkg-shlibdeps(1) dpkg suite dpkg-shlibdeps(1)

NAME

dpkg-shlibdeps - generate shared library substvar dependencies

SYNOPSIS

dpkg-shlibdeps [option...] [-e]executable [option...]

DESCRIPTION

dpkg-shlibdeps calculates shared library dependencies for executables named in its arguments. The dependencies are added to the substitution variables file debian/substvars as variable names shlibs:dependency-field where dependency-field is a dependency field name. Any other variables starting with shlibs: are removed from the file.

dpkg-shlibdeps has two possible sources of information to generate dependency information. Either symbols files or shlibs files. For each binary that dpkg-shlibdeps analyzes, it finds out the list of libraries that it's linked with. Then, for each library, it looks up either the symbols file, or the shlibs file (if the former doesn't exist or if debian/shlibs.local contains the relevant dependency). Both files are supposed to be provided by the library package and should thus be available as /var/lib/dpkg/info/package.symbols or /var/lib/dpkg/info/package.shlibs. The package name is identified in two steps: find the library file on the system (looking in the same directories that ld.so would use), then use dpkg -S library-file to lookup the package providing the library.

Symbols files

Symbols files contain finer-grained dependency information by providing the minimum dependency for each symbol that the library exports. The script tries to find a symbols file associated to a library package in the following places (first match is used):

debian/*/DEBIAN/symbols

Shared library information generated by the current build process that also invoked dpkg-shlibdeps. They are generated by dpkg-gensymbols(1). They are only used if the library is found in a package's build tree. The symbols file in that build tree takes precedence over symbols files from other binary packages.

/etc/dpkg/symbols/package.symbols.arch

/etc/dpkg/symbols/package.symbols

Per-system overriding shared library dependency information. arch is the architecture of the current system (obtained by dpkg-architecture -qDEB_HOST_ARCH).

Output from ?dpkg-query --control-path package symbols?

Package-provided shared library dependency information. Unless overridden by --admindir, those files are located in /var/lib/dpkg.

While scanning the symbols used by all binaries, dpkg-shlibdeps remembers the (biggest) minimal version needed for each library. At the end of the process, it is able to write out the minimal dependency for every library used (provided that the information of the symbols files are accurate).

As a safe-guard measure, a symbols file can provide a Build-Depends-Package meta-information field and dpkg-shlibdeps will extract the minimal version required by the corresponding package in the Build-Depends field and use this version if it's higher than the minimal version computed by scanning symbols.

Shlibs files

Shlibs files associate directly a library to a dependency (without looking at the symbols). It's thus often stronger than really needed but very safe and easy to handle.

The dependencies for a library are looked up in several places. The first file providing information for the library of interest is used:

debian/shlibs.local

Package-local overriding shared library dependency information.

/etc/dpkg/shlibs.override

Per-system overriding shared library dependency information.

debian/*/DEBIAN/shlibs

Shared library information generated by the current build process that also invoked dpkg-shlibdeps. They are only used if the library is found in a package's build tree. The shlibs file in that build tree takes precedence over shlibs files from other binary packages.

Output from ?dpkg-query --control-path package shlibs?

Package-provided shared library dependency information. Unless overridden by --admindir, those files are located in /var/lib/dpkg.

/etc/dpkg/shlibs.default

Per-system default shared library dependency information.

The extracted dependencies are then directly used (except if they are filtered out because they have been identified as duplicate, or as weaker than another dependency).

OPTIONS

dpkg-shlibdeps interprets non-option arguments as executable names, just as if they'd been supplied as -eexecutable.

-eexecutable

Include dependencies appropriate for the shared libraries required by executable. This option can be used multiple times.

-ldirectory

Prepend directory to the list of directories to search for private shared libraries (since dpkg 1.17.0). This option can be used multiple times.

Note: Use this option instead of setting LD_LIBRARY_PATH, as that environment variable is used to control the run-time linker and abusing it to set the shared library paths at build-time can be problematic when cross-compiling for example.

-ddependency-field

Add dependencies to be added to the control file dependency field dependency-field. (The dependencies for this field are placed in the

variable shlibs:dependency-field.)

The -ddependency-field option takes effect for all executables after the option, until the next -ddependency-field. The default dependency-field is Depends.

If the same dependency entry (or set of alternatives) appears in more than one of the recognized dependency field names Pre-Depends, Depends, Recommends, Enhances or Suggests then dpkg-shlibdeps will automatically remove the dependency from all fields except the one representing the most important dependencies.

-pvarname-prefix

Start substitution variables with varname-prefix: instead of shlibs:. Likewise, any existing substitution variables starting with varname-prefix: (rather than shlibs:) are removed from the substitution variables file.

-O[filename]

Print substitution variable settings to standard output (or filename if specified, since dpkg 1.17.2), rather than being added to the substitution variables file (debian/substvars by default).

-ttype Prefer shared library dependency information tagged for the given package type. If no tagged information is available, falls back to untagged information. The default package type is deb. Shared library dependency information is tagged for a given type by prefixing it with the name of the type, a colon, and whitespace.

-Llocal-shlibs-file

Read overriding shared library dependency information from local-shlibs-file instead of debian/shlibs.local.

-Tsubstvars-file

Write substitution variables in substvars-file; the default is debian/substvars.

-v Enable verbose mode (since dpkg 1.14.8). Numerous messages are displayed to explain what dpkg-shlibdeps does.

-xpackage

Exclude the package from the generated dependencies (since dpkg 1.14.8). This is useful to avoid self-dependencies for packages which provide ELF

binaries (executables or library plugins) using a library contained in the same package. This option can be used multiple times to exclude several packages.

-Spackage-build-dir

Look into package-build-dir first when trying to find a library (since dpkg 1.14.15). This is useful when the source package builds multiple flavors of the same library and you want to ensure that you get the dependency from a given binary package. You can use this option multiple times: directories will be tried in the same order before directories of other binary packages.

-Ipackage-build-dir

Ignore package-build-dir when looking for shlibs, symbols, and shared library files (since dpkg 1.18.5). You can use this option multiple times.

--ignore-missing-info

Do not fail if dependency information can't be found for a shared library (since dpkg 1.14.8). Usage of this option is discouraged, all libraries should provide dependency information (either with shlibs files, or with symbols files) even if they are not yet used by other packages.

--warnings=value

value is a bit field defining the set of warnings that can be emitted by dpkg-shlibdeps (since dpkg 1.14.17). Bit 0 (value=1) enables the warning ?symbol sym used by binary found in none of the libraries?, bit 1 (value=2) enables the warning ?package could avoid a useless dependency? and bit 2 (value=4) enables the warning ?binary should not be linked against library?. The default value is 3: the first two warnings are active by default, the last one is not. Set value to 7 if you want all warnings to be active.

--admindir=dir

Change the location of the dpkg database (since dpkg 1.14.0). The default location is /var/lib/dpkg.

-?, --help

Show the usage message and exit.

--version

Show the version and exit.

DPKG_COLORS

Sets the color mode (since dpkg 1.18.5). The currently accepted values are: auto (default), always and never.

DPKG_NLS

If set, it will be used to decide whether to activate Native Language Support, also known as internationalization (or i18n) support (since dpkg 1.19.0). The accepted values are: 0 and 1 (default).

## DIAGNOSTICS

### Warnings

Since dpkg-shlibdeps analyzes the set of symbols used by each binary of the generated package, it is able to emit warnings in several cases. They inform you of things that can be improved in the package. In most cases, those improvements concern the upstream sources directly. By order of decreasing importance, here are the various warnings that you can encounter:

symbol sym used by binary found in none of the libraries.

The indicated symbol has not been found in the libraries linked with the binary. The binary is most likely a library and it needs to be linked with an additional library during the build process (option -llibrary of the linker).

binary contains an unresolvable reference to symbol sym: it's probably a plugin

The indicated symbol has not been found in the libraries linked with the binary. The binary is most likely a plugin and the symbol is probably provided by the program that loads this plugin. In theory a plugin doesn't have any SONAME but this binary does have one and as such it could not be clearly identified as such. However the fact that the binary is stored in a non-public directory is a strong indication that's it's not a normal shared library. If the binary is really a plugin, then disregard this warning. But there's always the possibility that it's a real library and that programs linking to it are using an RPATH so that the dynamic loader finds it. In that case, the library is broken and needs to be fixed.

package could avoid a useless dependency if binary was not linked against library (it uses none of the library's symbols)

None of the binaries that are linked with library use any of the symbols

provided by the library. By fixing all the binaries, you would avoid the dependency associated to this library (unless the same dependency is also generated by another library that is really used).

package could avoid a useless dependency if binaries were not linked against library (they use none of the library's symbols)

Exactly the same as the above warning, but for multiple binaries.

binary should not be linked against library (it uses none of the library's symbols)

The binary is linked to a library that it doesn't need. It's not a problem but some small performance improvements in binary load time can be obtained by not linking this library to this binary. This warning checks the same information as the previous one but does it for each binary instead of doing the check globally on all binaries analyzed.

Errors

dpkg-shlibdeps will fail if it can't find a public library used by a binary or if this library has no associated dependency information (either shlibs file or symbols file). A public library has a SONAME and is versioned (libsomething.so.X). A private library (like a plugin) should not have a SONAME and doesn't need to be versioned.

couldn't find library library-soname needed by binary (its RPATH is 'rpath')

The binary uses a library called library-soname but dpkg-shlibdeps has been unable to find the library. dpkg-shlibdeps creates a list of directories to check as following: directories listed in the RPATH of the binary, directories added by the -l option, directories listed in the LD_LIBRARY_PATH environment variable, cross multiarch directories (ex. /lib/arm64-linux-gnu, /usr/lib/arm64-linux-gnu), standard public directories (/lib, /usr/lib), directories listed in /etc/ld.so.conf, and obsolete multilib directories (/lib32, /usr/lib32, /lib64, /usr/lib64). Then it checks those directories in the package's build tree of the binary being analyzed, in the packages' build trees indicated with the -S command-line option, in other packages' build trees that contains a DEBIAN/shlibs or DEBIAN/symbols file and finally in the root directory. If the library is not found in any of those directories, then you get this error.

If the library not found is in a private directory of the same package, then

you want to add the directory with -l. If it's in another binary package being built, you want to make sure that the shlibs/symbols file of this package is already created and that -l contains the appropriate directory if it also is in a private directory.

no dependency information found for library-file (used by binary).

The library needed by binary has been found by dpkg-shlibdeps in library-file but dpkg-shlibdeps has been unable to find any dependency information for that library. To find out the dependency, it has tried to map the library to a Debian package with the help of dpkg -S library-file. Then it checked the corresponding shlibs and symbols files in /var/lib/dpkg/info/, and in the various package's build trees (debian/*/DEBIAN/).

This failure can be caused by a bad or missing shlibs or symbols file in the package of the library. It might also happen if the library is built within the same source package and if the shlibs files has not yet been created (in which case you must fix debian/rules to create the shlibs before calling dpkg-shlibdeps). Bad RPATH can also lead to the library being found under a non-canonical name (example: /usr/lib/openoffice.org/../lib/libssl.so.0.9.8 instead of /usr/lib/libssl.so.0.9.8) that's not associated to any package, dpkg-shlibdeps tries to work around this by trying to fallback on a canonical name (using realpath(3)) but it might not always work. It's always best to clean up the RPATH of the binary to avoid problems.

Calling dpkg-shlibdeps in verbose mode (-v) will provide much more information about where it tried to find the dependency information. This might be useful if you don't understand why it's giving you this error.

## SEE ALSO

deb-shlibs(5), deb-symbols(5), dpkg-gensymbols(1).