Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'c99.1p' command

$ man c99.1p

C99(1P) POSIX Programmer's Manual C99(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

c99 ? compile standard C programs

SYNOPSIS

c99 [options...] pathname [[pathname] [-I directory] [-L directory] [-l library]]...

DESCRIPTION

The c99 utility is an interface to the standard C compilation system; it shall accept source code conforming to the ISO C standard. The sys? tem conceptually consists of a compiler and link editor. The input files referenced by pathname operands and -l option-arguments shall be compiled and linked to produce an executable file. (It is unspecified whether the linking occurs entirely within the operation of c99; some implementations may produce objects that are not fully resolved until the file is executed.)

If the -c option is specified, for all pathname operands of the form file.c, the files:

$(basename pathname .c).o

shall be created as the result of successful compilation. If the -c op?
tion is not specified, it is unspecified whether such .o files are cre?
ated or deleted for the file.c operands.

If there are no options that prevent link editing (such as -c or -E),
and all input files compile and link without error, the resulting exe?
cutable file shall be written according to the -o outfile option (if
present) or to the file a.out.

The executable file shall be created as specified in Section 1.1.1.4,
File Read, Write, and Creation, except that the file permission bits
shall be set to: S_IRWXO | S_IRWXG | S_IRWXU
and the bits specified by the umask of the process shall be cleared.

OPTIONS

The c99 utility shall conform to the Base Definitions volume of
POSIX.1?2017, Section 12.2, Utility Syntax Guidelines, except that:

* Options can be interspersed with operands.
* The order of specifying the -L and -l options, and the order of
  specifying -l options with respect to pathname operands is signifi?
  cant.
* Conforming applications shall specify each option separately; that
  is, grouping option letters (for example, -cO) need not be recog?
  nized by all implementations.

The following options shall be supported:

-c Suppress the link-edit phase of the compilation, and do not
remove any object files that are produced.

-D name[=value]

Define name as if by a C-language #define directive. If no
=value is given, a value of 1 shall be used. The -D option
has lower precedence than the -U option. That is, if name is
used in both a -U and a -D option, name shall be undefined
regardless of the order of the options. Additional implemen?
tation-defined names may be provided by the compiler. Imple?
mentations shall support at least 2048 bytes of -D defini?
tions and 256 names.

-E Copy C-language source files to standard output, executing all preprocessor directives; no compilation shall be per? formed. If any operand is not a text file, the effects are unspecified.

-g Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.

-I directory

Change the algorithm for searching for headers whose names are not absolute pathnames to look in the directory named by the directory pathname before looking in the usual places. Thus, headers whose names are enclosed in double-quotes ("") shall be searched for first in the directory of the file with the #include line, then in directories named in -I options, and last in the usual places. For headers whose names are en? closed in angle brackets ("<>"), the header shall be searched for only in directories named in -I options and then in the usual places. Directories named in -I options shall be searched in the order specified. If the -I option is used to specify a directory that is one of the usual places searched by default, the results are unspecified. Implementations shall support at least ten instances of this option in a sin? gle c99 command invocation.

-L directory

Change the algorithm of searching for the libraries named in the -l objects to look in the directory named by the direc? tory pathname before looking in the usual places. Directories named in -L options shall be searched in the order specified. If the -L option is used to specify a directory that is one of the usual places searched by default, the results are un? specified. Implementations shall support at least ten in? stances of this option in a single c99 command invocation. If

a directory specified by a -L option contains files with names starting with any of the strings "libc.", "libl.", "libpthread.", "libm.", "librt.", "libtrace.", "libxnet.", or "liby.", the results are unspecified.

-l library

Search the library named liblibrary.a. A library shall be searched when its name is encountered, so the placement of a -l option is significant. Several standard libraries can be specified in this manner, as described in the EXTENDED DE? SCRIPTION section. Implementations may recognize implementa? tion-defined suffixes other than .a as denoting libraries.

-O optlevel

Specify the level of code optimization. If the optlevel op? tion-argument is the digit '0', all special code optimiza? tions shall be disabled. If it is the digit '1', the nature of the optimization is unspecified. If the -O option is omit? ted, the nature of the system's default optimization is un? specified. It is unspecified whether code generated in the presence of the -O 0 option is the same as that generated when -O is omitted. Other optlevel values may be supported.

-o outfile

Use the pathname outfile, instead of the default a.out, for the executable file produced. If the -o option is present with -c or -E, the result is unspecified.

-s Produce object or executable files, or both, from which sym? bolic and other information not required for proper execution using the exec family defined in the System Interfaces volume of POSIX.1?2017 has been removed (stripped). If both -g and -s options are present, the action taken is unspecified.

-U name Remove any initial definition of name.

Multiple instances of the -D, -I, -L, -l, and -U options can be speci? fied.

OPERANDS

The application shall ensure that at least one pathname operand is specified. The following forms for pathname operands shall be sup? ported:

file.c A C-language source file to be compiled and optionally linked. The application shall ensure that the operand is of this form if the -c option is used.

file.a A library of object files typically produced by the ar util? ity, and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than .a as denoting object file libraries.

file.o An object file produced by c99 -c and passed directly to the link editor. Implementations may recognize implementation-de? fined suffixes other than .o as denoting object files.

The processing of other files is implementation-defined.

STDIN

Not used.

INPUT FILES

Each input file shall be one of the following: a text file containing a C-language source program, an object file in the format produced by c99 -c, or a library of object files, in the format produced by archiving zero or more object files, using ar. Implementations may supply addi? tional utilities that produce files in these formats. Additional input file formats are implementation-defined.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of c99:

LANG Provide a default value for the internationalization vari? ables that are unset or null. (See the Base Definitions vol? ume of POSIX.1?2017, Section 8.2, Internationalization Vari? ables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of

bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard er? ror.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

TMPDIR Provide a pathname that should override the default directory for temporary files, if any. On XSI-conforming systems, pro? vide a pathname that shall override the default directory for temporary files, if any.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If more than one pathname operand ending in .c (or possibly other un? specified suffixes) is given, for each such file:

"%s:\n", <pathname>

may be written. These messages, if written, shall precede the process? ing of each input file; they shall not be written to the standard out? put if they are written to the standard error, as described in the STDERR section.

If the -E option is specified, the standard output shall be a text file that represents the results of the preprocessing stage of the language; it may contain extra information appropriate for subsequent compilation passes.

STDERR

The standard error shall be used only for diagnostic messages. If more than one pathname operand ending in .c (or possibly other unspecified suffixes) is given, for each such file:

"%s:\n", <pathname>

may be written to allow identification of the diagnostic and warning

messages with the appropriate input file. These messages, if written, shall precede the processing of each input file; they shall not be written to the standard error if they are written to the standard out? put, as described in the STDOUT section.

This utility may produce warning messages about certain conditions that do not warrant returning an error (non-zero) exit value.

OUTPUT FILES

Object files or executable files or both are produced in unspecified formats. If the pathname of an object file or executable file to be created by c99 resolves to an existing directory entry for a file that is not a regular file, it is unspecified whether c99 shall attempt to create the file or shall issue a diagnostic and exit with a non-zero exit status.

EXTENDED DESCRIPTION

Standard Libraries

The c99 utility shall recognize the following -l options for standard libraries:

-l c This option shall make available all interfaces referenced in the System Interfaces volume of POSIX.1?2017, with the possi? ble exception of those interfaces listed as residing in <aio.h>, <arpa/inet.h>, <complex.h>, <fenv.h>, <math.h>, <mqueue.h>, <netdb.h>, <net/if.h>, <netinet/in.h>, <pthread.h>, <sched.h>, <semaphore.h>, <spawn.h>, <sys/socket.h>, pthread_kill(), and pthread_sigmask() in <signal.h>, <trace.h>, interfaces marked as optional in <sys/mman.h>, interfaces marked as ADV (Advisory Information) in <fcntl.h>, and interfaces beginning with the prefix clock_ or timer_ in <time.h>. This option shall not be required to be present to cause a search of this library.

-l l This option shall make available all interfaces required by the C-language output of lex that are not made available through the -l c option.

-l pthread

This option shall make available all interfaces referenced in <pthread.h> and pthread_kill() and pthread_sigmask() refer? enced in <signal.h>. An implementation may search this li? brary in the absence of this option.

-l m This option shall make available all interfaces referenced in <math.h>, <complex.h>, and <fenv.h>. An implementation may search this library in the absence of this option.

-l rt This option shall make available all interfaces referenced in <aio.h>, <mqueue.h>, <sched.h>, <semaphore.h>, and <spawn.h>, interfaces marked as optional in <sys/mman.h>, interfaces marked as ADV (Advisory Information) in <fcntl.h>, and inter? faces beginning with the prefix clock_ and timer_ in <time.h>. An implementation may search this library in the absence of this option.

-l trace This option shall make available all interfaces referenced in <trace.h>. An implementation may search this library in the absence of this option.

-l xnet This option shall make available all interfaces referenced in <arpa/inet.h>, <netdb.h>, <net/if.h>, <netinet/in.h>, and <sys/socket.h>. An implementation may search this library in the absence of this option.

-l y This option shall make available all interfaces required by the C-language output of yacc that are not made available through the -l c option.

In the absence of options that inhibit invocation of the link editor, such as -c or -E, the c99 utility shall cause the equivalent of a -l c option to be passed to the link editor after the last pathname operand or -l option, causing it to be searched after all other object files and libraries are loaded.

It is unspecified whether the libraries libc.a, libl.a, libm.a, libpthread.a, librt.a, libtrace.a, libxnet.a, or liby.a exist as regu? lar files. The implementation may accept as -l option-arguments names of objects that do not exist as regular files.

External Symbols

The C compiler and link editor shall support the significance of exter?
nal symbols up to a length of at least 31 bytes; the action taken upon
encountering symbols exceeding the implementation-defined maximum sym?
bol length is unspecified.

The compiler and link editor shall support a minimum of 511 external
symbols per source or object file, and a minimum of 4095 external sym?
bols in total. A diagnostic message shall be written to the standard
output if the implementation-defined limit is exceeded; other actions
are unspecified.

Header Search

If a file with the same name as one of the standard headers defined in
the Base Definitions volume of POSIX.1?2017, Chapter 13, Headers, not
provided as part of the implementation, is placed in any of the usual
places that are searched by default for headers, the results are un?
specified.

Programming Environments

All implementations shall support one of the following programming en?
vironments as a default. Implementations may support more than one of
the following programming environments. Applications can use sysconf()
or getconf to determine which programming environments are supported.

Table 4-4: Programming Environments: Type Sizes

Programming Environment	Bits in int	Bits in long	Bits in pointer	Bits in off_t
getconf Name				
_POSIX_V7_ILP32_OFF32	32	32	32	32
_POSIX_V7_ILP32_OFFBIG	32	32	32	?64
_POSIX_V7_LP64_OFF64	32	64	64	64
_POSIX_V7_LPBIG_OFFBIG	?32	?64	?64	?64

All implementations shall support one or more environments where the
widths of the following types are no greater than the width of type

long:

blksize_t ptrdiff_t tcflag_t

cc_t size_t wchar_t

mode_t speed_t wint_t

nfds_t ssize_t

pid_t suseconds_t

The executable files created when these environments are selected shall be in a proper format for execution by the exec family of functions.

Each environment may be one of the ones in Table 4-4, Programming Envi? ronments: Type Sizes, or it may be another environment. The names for the environments that meet this requirement shall be output by a get? conf command using the POSIX_V7_WIDTH_RESTRICTED_ENVS argument, as a <newline>-separated list of names suitable for use with the getconf -v option. If more than one environment meets the requirement, the names of all such environments shall be output on separate lines. Any of these names can then be used in a subsequent getconf command to obtain the flags specific to that environment with the following suffixes added as appropriate:

_CFLAGS To get the C compiler flags.

_LDFLAGS To get the linker/loader flags.

_LIBS To get the libraries.

This requirement may be removed in a future version.

When this utility processes a file containing a function called main(), it shall be defined with a return type equivalent to int. Using return from the initial call to main() shall be equivalent (other than with respect to language scope issues) to calling exit() with the returned value. Reaching the end of the initial call to main() shall be equiva? lent to calling exit(0). The implementation shall not declare a proto? type for this function.

Implementations provide configuration strings for C compiler flags, linker/loader flags, and libraries for each supported environment.

When an application needs to use a specific programming environment rather than the implementation default programming environment while

compiling, the application shall first verify that the implementation supports the desired environment. If the desired programming environ? ment is supported, the application shall then invoke c99 with the ap? propriate C compiler flags as the first options for the compile, the appropriate linker/loader flags after any other options except -l but before any operands or -l options, and the appropriate libraries at the end of the operands and -l options.

Conforming applications shall not attempt to link together object files compiled for different programming models. Applications shall also be aware that binary data placed in shared memory or in files might not be recognized by applications built for other programming models.

Table 4-5: Programming Environments: c99 Arguments

Programming Environment		c99 Arguments	
getconf Name	Use	getconf Name	
_POSIX_V7_ILP32_OFF32	C Compiler Flags	POSIX_V7_ILP32_OFF32_CFLAGS	
	Linker/Loader Flags	POSIX_V7_ILP32_OFF32_LDFLAGS	
	Libraries	POSIX_V7_ILP32_OFF32_LIBS	
_POSIX_V7_ILP32_OFFBIG	C Compiler Flags	POSIX_V7_ILP32_OFFBIG_CFLAGS	
	Linker/Loader Flags	POSIX_V7_ILP32_OFFBIG_LDFLAGS	
	Libraries	POSIX_V7_ILP32_OFFBIG_LIBS	
_POSIX_V7_LP64_OFF64	C Compiler Flags	POSIX_V7_LP64_OFF64_CFLAGS	
	Linker/Loader Flags	POSIX_V7_LP64_OFF64_LDFLAGS	
	Libraries	POSIX_V7_LP64_OFF64_LIBS	
_POSIX_V7_LPBIG_OFFBIG	C Compiler Flags	POSIX_V7_LPBIG_OFFBIG_CFLAGS	
	Linker/Loader Flags	POSIX_V7_LPBIG_OFFBIG_LDFLAGS	
	Libraries	POSIX_V7_LPBIG_OFFBIG_LIBS	

In addition to the type size programming environments above, all imple?

mentations also support a multi-threaded programming environment that is orthogonal to all of the programming environments listed above. The getconf utility can be used to get flags for the threaded programming environment, as indicated in Table 4-6, Threaded Programming Environ? ment: c99 Arguments.

Table 4-6: Threaded Programming Environment: c99 Arguments

Programming Environment		c99 Arguments	
getconf Name	Use	getconf Name	
_POSIX_THREADS	C Compiler Flags	POSIX_V7_THREADS_CFLAGS	
	Linker/Loader Flags	POSIX_V7_THREADS_LDFLAGS	

These programming environment flags may be used in conjunction with any of the type size programming environments supported by the implementa? tion.

EXIT STATUS

The following exit values shall be returned:

0 Successful compilation or link edit.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When c99 encounters a compilation error that causes an object file not to be created, it shall write a diagnostic to standard error and con? tinue to compile other source code operands, but it shall not perform the link phase and it shall return a non-zero exit status. If the link edit is unsuccessful, a diagnostic message shall be written to standard error and c99 exits with a non-zero status. A conforming application shall rely on the exit status of c99, rather than on the existence or mode of the executable file.

The following sections are informative.

APPLICATION USAGE

Since the c99 utility usually creates files in the current directory during the compilation process, it is typically necessary to run the

c99 utility in a directory in which a file can be created.

On systems providing POSIX Conformance (see the Base Definitions volume of POSIX.1?2017, Chapter 2, Conformance), c99 is required only with the C-Language Development option; XSI-conformant systems always provide c99.

Some historical implementations have created .o files when -c is not specified and more than one source file is given. Since this area is left unspecified, the application cannot rely on .o files being cre? ated, but it also must be prepared for any related .o files that al? ready exist being deleted at the completion of the link edit.

There is the possible implication that if a user supplies versions of the standard functions (before they would be encountered by an implicit -l c or explicit -l m), that those versions would be used in place of the standard versions. There are various reasons this might not be true (functions defined as macros, manipulations for clean name space, and so on), so the existence of files named in the same manner as the standard libraries within the -L directories is explicitly stated to produce unspecified behavior.

All of the functions specified in the System Interfaces volume of POSIX.1?2017 may be made visible by implementations when the Standard C Library is searched. Conforming applications must explicitly request searching the other standard libraries when functions made visible by those libraries are used.

In the ISO C standard the mapping from physical source characters to the C source character set is implementation-defined. Implementations may strip white-space characters before the terminating <newline> of a (physical) line as part of this mapping and, as a consequence of this, one or more white-space characters (and no other characters) between a <backslash> character and the <newline> character that terminates the line produces implementation-defined results. Portable applications should not use such constructs.

Some c99 compilers not conforming to POSIX.1?2008 do not support tri? graphs by default.

EXAMPLES

1. The following usage example compiles foo.c and creates the exe?

   cutable file foo:

   ```
   c99 -o foo foo.c
   ```

   The following usage example compiles foo.c and creates the object

   file foo.o:

   ```
   c99 -c foo.c
   ```

   The following usage example compiles foo.c and creates the exe?

   cutable file a.out:

   ```
   c99 foo.c
   ```

   The following usage example compiles foo.c, links it with bar.o,

   and creates the executable file a.out. It may also create and

   leave foo.o:

   ```
   c99 foo.c bar.o
   ```

2. The following example shows how an application using threads inter?

   faces can test for support of and use a programming environment

   supporting 32-bit int, long, and pointer types and an off_t type

   using at least 64 bits:

   ```
   offbig_env=$(getconf _POSIX_V7_ILP32_OFFBIG)
   if [ $offbig_env != "-1" ] && [ $offbig_env != "undefined" ]
   then
       c99 $(getconf POSIX_V7_ILP32_OFFBIG_CFLAGS) \
       $(getconf POSIX_V7_THREADS_CFLAGS) -D_XOPEN_SOURCE=700 \
       $(getconf POSIX_V7_ILP32_OFFBIG_LDFLAGS) \
       $(getconf POSIX_V7_THREADS_LDFLAGS) foo.c -o foo \
       $(getconf POSIX_V7_ILP32_OFFBIG_LIBS) \
       -l pthread
   else
       echo ILP32_OFFBIG programming environment not supported
       exit 1
   fi
   ```

3. The following examples clarify the use and interactions of -L and

   -l options.

Consider the case in which module a.c calls function f() in library libQ.a, and module b.c calls function g() in library libp.a. As? sume that both libraries reside in /a/b/c. The command line to compile and link in the desired way is:

```
c99 -L /a/b/c main.o a.c -l Q b.c -l p
```

In this case the -L option need only precede the first -l option, since both libQ.a and libp.a reside in the same directory.

Multiple -L options can be used when library name collisions occur. Building on the previous example, suppose that the user wants to use a new libp.a, in /a/a/a, but still wants f() from /a/b/c/libQ.a:

```
c99 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

In this example, the linker searches the -L options in the order specified, and finds /a/a/a/libp.a before /a/b/c/libp.a when re? solving references for b.c. The order of the -l options is still important, however.

4. The following example shows how an application can use a program? ming environment where the widths of the following types: blk? size_t, cc_t, mode_t, nfds_t, pid_t, ptrdiff_t, size_t, speed_t, ssize_t, suseconds_t, tcflag_t, wchar_t, wint_t are no greater than the width of type long:

```
# First choose one of the listed environments ...
# ... if there are no additional constraints, the first one will do:
CENV=$(getconf POSIX_V7_WIDTH_RESTRICTED_ENVS | head -n l)
# ... or, if an environment that supports large files is preferred,
# look for names that contain "OFF64" or "OFFBIG". (This chooses
# the last one in the list if none match.)
for CENV in $(getconf POSIX_V7_WIDTH_RESTRICTED_ENVS)
do
  case $CENV in
  *OFF64*|*OFFBIG*) break ;;
  esac
done
```

```
# The chosen environment name can now be used like this:
c99 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=200809L \
$(getconf ${CENV}_LDFLAGS) foo.c -o foo \
$(getconf ${CENV}_LIBS)
```

RATIONALE

The c99 utility is based on the c89 utility originally introduced in the ISO POSIX?2:1993 standard.

Some of the changes from c89 include the ability to intersperse options and operands (which many c89 implementations allowed despite it not be? ing specified), the description of -l as an option instead of an oper? and, and the modification to the contents of the Standard Libraries section to account for new headers and options; for example, <spawn.h> added to the description of -l rt, and -l trace added for the Tracing option.

POSIX.1?2008 specifies that the c99 utility must be able to use regular files for *.o files and for a.out files. Implementations are free to overwrite existing files of other types when attempting to create ob? ject files and executable files, but are not required to do so. If something other than a regular file is specified and using it fails for any reason, c99 is required to issue a diagnostic message and exit with a non-zero exit status. But for some file types, the problem may not be noticed for a long time. For example, if a FIFO named a.out exists in the current directory, c99 may attempt to open a.out and will hang in the open() call until another process opens the FIFO for reading. Then c99 may write most of the a.out to the FIFO and fail when it tries to seek back close to the start of the file to insert a timestamp (FIFOs are not seekable files). The c99 utility is also allowed to issue a di? agnostic immediately if it encounters an a.out or *.o file that is not a regular file. For portable use, applications should ensure that any a.out, -o option-argument, or *.o files corresponding to any *.c files do not conflict with names already in use that are not regular files or symbolic links that point to regular files.

On many systems, multi-threaded applications run in a programming envi?

ronment that is distinct from that used by single-threaded applica? tions. This multi-threaded programming environment (in addition to needing to specify -l pthread at link time) may require additional flags to be set when headers are processed at compile time (-D_REEN? TRANT being common). This programming environment is orthogonal to the type size programming environments discussed above and listed in Table 4-4, Programming Environments: Type Sizes. This version of the stan? dard adds getconf utility calls to provide the C compiler flags and linker/loader flags needed to support multi-threaded applications. Note that on a system where single-threaded applications are a special case of a multi-threaded application, both of these getconf calls may return NULL strings; on other implementations both of these strings may be non-NULL strings.

The C standardization committee invented trigraphs (e.g., "??!" to rep? resent '|') to address character portability problems in development environments based on national variants of the 7-bit ISO/IEC 646:1991 standard character set. However, these environments were already obso? lete by the time the first ISO C standard was published, and in prac? tice trigraphs have not been used for their intended purpose, and usu? ally are intended to have their original meaning in K&R C. For exam? ple, in practice a C-language source string like "What??!" is usually intended to end in two <question-mark> characters and an <exclamation-mark>, not in '|'.

When the -E option is used, execution of some #pragma preprocessor di? rectives may simply result in a copy of the directive being included in the output as part of the allowed extra information used by subsequent compilation passes (see STDOUT).

FUTURE DIRECTIONS

Unlike all of the other non-OB-shaded utilities in this standard, a utility by this name probably will not appear in the next version of this standard. This utility's name is tied to the current revision of the ISO C standard at the time this standard is approved. Since the ISO C standard and this standard are maintained by different organiza?

tions on different schedules, we cannot predict what the compiler will be named in the next version of the standard.

SEE ALSO

Section 1.1.1.4, File Read, Write, and Creation, ar, getconf, make, nm, strip, umask

The Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables, Section 12.2, Utility Syntax Guidelines, Chapter 13, Headers

The System Interfaces volume of POSIX.1?2017, exec, sysconf()

COPYRIGHT