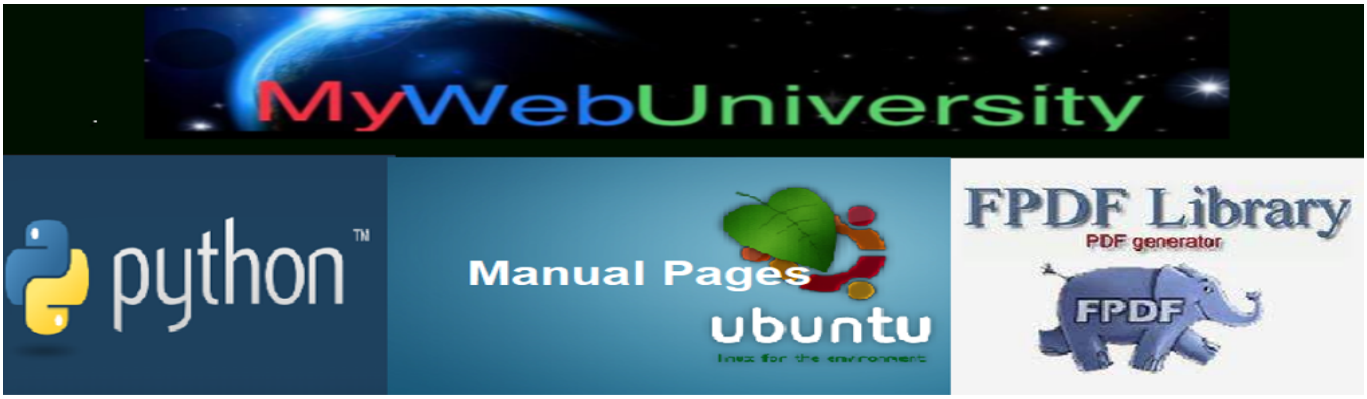




## *Rocky Enterprise Linux 9.2 Manual Pages on command 'cpp.1'*

***$ man cpp.1***

CPP(1) GNU CPP(1)

NAME

cpp - The C Preprocessor

SYNOPSIS

cpp [-Dmacro[=defn]...] [-Umacro]

[-Idir...] [-iquotedir...]

[-M|-MM] [-MG] [-MF filename]

[-MP] [-MQ target...]

[-MT target...]

infile [[-o] outfile]

Only the most useful options are given above; see below for a more complete list of preprocessor-specific options. In addition, cpp accepts most gcc driver options, which are not listed here. Refer to the GCC documentation for details.

DESCRIPTION

The C preprocessor, often known as cpp, is a macro processor that is used automatically by the C compiler to transform your program before compilation. It is called a macro processor because it allows you to define macros, which are brief abbreviations for longer constructs.

The C preprocessor is intended to be used only with C, C++, and Objective-C source code. In the past, it has been abused as a general text processor. It will choke on input which does not obey C's lexical rules. For example, apostrophes will be interpreted as the beginning of character constants, and cause errors. Also, you cannot rely on it preserving characteristics of the input which are not significant to C-family languages.

If a Makefile is preprocessed, all the hard tabs will be removed, and the Makefile will not work.

Having said that, you can often get away with using cpp on things which are not C. Other Algol-ish programming languages are often safe (Ada, etc.) So is assembly, with caution. -traditional-cpp mode preserves more white space, and is otherwise more permissive. Many of the problems can be avoided by writing C or C++ style comments instead of native language comments, and keeping macros simple.

Wherever possible, you should use a preprocessor geared to the language you are writing in. Modern versions of the GNU assembler have macro facilities. Most high level programming languages have their own conditional compilation and inclusion mechanism. If all else fails, try a true general text processor, such as GNU M4.

C preprocessors vary in some details. This manual discusses the GNU C preprocessor, which provides a small superset of the features of ISO Standard C. In its default mode, the GNU C preprocessor does not do a few things required by the standard. These are features which are rarely, if ever, used, and may cause surprising changes to the meaning of a program which does not expect them. To get strict ISO Standard C, you should use the -std=c90, -std=c99, -std=c11 or -std=c17 options, depending on which version of the standard you want. To get all the mandatory diagnostics, you must also use -pedantic.

This manual describes the behavior of the ISO preprocessor. To minimize gratuitous differences, where the ISO preprocessor's behavior does not conflict with traditional semantics, the traditional preprocessor should behave the same way. The various differences that do exist are detailed in the section Traditional Mode.

For clarity, unless noted otherwise, references to CPP in this manual refer to GNU CPP.

OPTIONS

The cpp command expects two file names as arguments, infile and outfile. The preprocessor reads infile together with any other files it specifies with #include. All the output generated by the combined input files is written in outfile.

Either infile or outfile may be -, which as infile means to read from standard input and as outfile means to write to standard output. If either file is omitted, it means the same as if - had been specified for that file. You can also use the -o outfile option to specify the output file.

Unless otherwise noted, or the option ends in =, all options which take an argument may have that argument appear either immediately after the option, or with a space between

option and argument: -Ifoo and -I foo have the same effect.

Many options have multi-letter names; therefore multiple single-letter options may not be grouped: -dM is very different from -d -M.

-D name

Predefine name as a macro, with definition 1.

-D name=definition

The contents of definition are tokenized and processed as if they appeared during translation phase three in a #define directive. In particular, the definition is truncated by embedded newline characters.

If you are invoking the preprocessor from a shell or shell-like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax.

If you wish to define a function-like macro on the command line, write its argument list with surrounding parentheses before the equals sign (if any). Parentheses are meaningful to most shells, so you should quote the option. With sh and csh, -D'name(args...)=definition' works.

-D and -U options are processed in the order they are given on the command line. All -imacros file and -include file options are processed after all -D and -U options.

-U name

Cancel any previous definition of name, either built in or provided with a -D option.

-include file

Process file as if "#include "file"" appeared as the first line of the primary source file. However, the first directory searched for file is the preprocessor's working directory instead of the directory containing the main source file. If not found there, it is searched for in the remainder of the "#include "..."" search chain as normal.

If multiple -include options are given, the files are included in the order they appear on the command line.

-imacros file

Exactly like -include, except that any output produced by scanning file is thrown away. Macros it defines remain defined. This allows you to acquire all the macros from a header without also processing its declarations.

All files specified by -imacros are processed before all files specified by -include.

-undef

Do not predefine any system-specific or GCC-specific macros. The standard predefined macros remain defined.

-pthread

Define additional macros required for using the POSIX threads library. You should use this option consistently for both compilation and linking. This option is supported on GNU/Linux targets, most other Unix derivatives, and also on x86 Cygwin and MinGW targets.

-M Instead of outputting the result of preprocessing, output a rule suitable for make describing the dependencies of the main source file. The preprocessor outputs one make rule containing the object file name for that source file, a colon, and the names of all the included files, including those coming from -include or -imacros command-line options.

Unless specified explicitly (with -MT or -MQ), the object file name consists of the name of the source file with any suffix replaced with object file suffix and with any leading directory parts removed. If there are many included files then the rule is split into several lines using \-newline. The rule has no commands.

This option does not suppress the preprocessor's debug output, such as -dM. To avoid mixing such debug output with the dependency rules you should explicitly specify the dependency output file with -MF, or use an environment variable like DEPENDENCIES_OUTPUT. Debug output is still sent to the regular output stream as normal.

Passing -M to the driver implies -E, and suppresses warnings with an implicit -w.

-MM Like -M but do not mention header files that are found in system header directories, nor header files that are included, directly or indirectly, from such a header.

This implies that the choice of angle brackets or double quotes in an #include directive does not in itself determine whether that header appears in -MM dependency output.

-MF file

When used with -M or -MM, specifies a file to write the dependencies to. If no -MF switch is given the preprocessor sends the rules to the same place it would send preprocessed output.

When used with the driver options -MD or -MMD, -MF overrides the default dependency

output file.

If file is -, then the dependencies are written to stdout.

-MG In conjunction with an option such as -M requesting dependency generation, -MG assumes missing header files are generated files and adds them to the dependency list without raising an error. The dependency filename is taken directly from the "#include" directive without prepending any path. -MG also suppresses preprocessed output, as a missing header file renders this useless.

This feature is used in automatic updating of makefiles.

-Mno-modules

Disable dependency generation for compiled module interfaces.

-MP This option instructs CPP to add a phony target for each dependency other than the main file, causing each to depend on nothing. These dummy rules work around errors make gives if you remove header files without updating the Makefile to match.

This is typical output:

```
test.o: test.c test.h
test.h:
```

-MT target

Change the target of the rule emitted by dependency generation. By default CPP takes the name of the main input file, deletes any directory components and any file suffix such as .c, and appends the platform's usual object suffix. The result is the target.

An -MT option sets the target to be exactly the string you specify. If you want multiple targets, you can specify them as a single argument to -MT, or use multiple -MT options.

For example, -MT '$(objpfx)foo.o' might give

```
$(objpfx)foo.o: foo.c
```

-MQ target

Same as -MT, but it quotes any characters which are special to Make.

-MQ '$(objpfx)foo.o' gives

```
$$(objpfx)foo.o: foo.c
```

The default target is automatically quoted, as if it were given with -MQ.

-MD -MD is equivalent to -M -MF file, except that -E is not implied. The driver determines file based on whether an -o option is given. If it is, the driver uses its argument but with a suffix of .d, otherwise it takes the name of the input file,

removes any directory components and suffix, and applies a .d suffix.

If -MD is used in conjunction with -E, any -o switch is understood to specify the dependency output file, but if used without -E, each -o is understood to specify a target object file.

Since -E is not implied, -MD can be used to generate a dependency output file as a side effect of the compilation process.

-MMD

Like -MD except mention only user header files, not system header files.

-fpreprocessed

Indicate to the preprocessor that the input file has already been preprocessed. This suppresses things like macro expansion, trigraph conversion, escaped newline splicing, and processing of most directives. The preprocessor still recognizes and removes comments, so that you can pass a file preprocessed with -C to the compiler without problems. In this mode the integrated preprocessor is little more than a tokenizer for the front ends.

-fpreprocessed is implicit if the input file has one of the extensions .i, .ii or .mi.

These are the extensions that GCC uses for preprocessed files created by -save-temps.

-fdirectives-only

When preprocessing, handle directives, but do not expand macros.

The option's behavior depends on the -E and -fpreprocessed options.

With -E, preprocessing is limited to the handling of directives such as "#define", "#ifdef", and "#error". Other preprocessor operations, such as macro expansion and trigraph conversion are not performed. In addition, the -dD option is implicitly enabled.

With -fpreprocessed, predefinition of command line and most builtin macros is disabled. Macros such as "__LINE__", which are contextually dependent, are handled normally. This enables compilation of files previously preprocessed with "-E -fdirectives-only".

With both -E and -fpreprocessed, the rules for -fpreprocessed take precedence. This enables full preprocessing of files previously preprocessed with "-E -fdirectives-only".

-fdollars-in-identifiers

Accept $ in identifiers.

-fextended-identifiers

Accept universal character names and extended characters in identifiers. This option is enabled by default for C99 (and later C standard versions) and C++.

-fno-canonical-system-headers

When preprocessing, do not shorten system header paths with canonicalization.

-fmax-include-depth=depth

Set the maximum depth of the nested #include. The default is 200.

-ftabstop=width

Set the distance between tab stops. This helps the preprocessor report correct column numbers in warnings or errors, even if tabs appear on the line. If the value is less than 1 or greater than 100, the option is ignored. The default is 8.

-ftrack-macro-expansion[=level]

Track locations of tokens across macro expansions. This allows the compiler to emit diagnostic about the current macro expansion stack when a compilation error occurs in a macro expansion. Using this option makes the preprocessor and the compiler consume more memory. The level parameter can be used to choose the level of precision of token location tracking thus decreasing the memory consumption if necessary. Value 0 of level de-activates this option. Value 1 tracks tokens locations in a degraded mode for the sake of minimal memory overhead. In this mode all tokens resulting from the expansion of an argument of a function-like macro have the same location. Value 2 tracks tokens locations completely. This value is the most memory hungry. When this option is given no argument, the default parameter value is 2.

Note that "-ftrack-macro-expansion=2" is activated by default.

-fmacro-prefix-map=old=new

When preprocessing files residing in directory old, expand the "__FILE__" and "__BASE_FILE__" macros as if the files resided in directory new instead. This can be used to change an absolute path to a relative path by using . for new which can result in more reproducible builds that are location independent. This option also affects "__builtin_FILE()" during compilation. See also -ffile-prefix-map.

-fexec-charset=charset

Set the execution character set, used for string and character constants. The default is UTF-8. charset can be any encoding supported by the system's "iconv" library routine.

-fwide-exec-charset=charset

Set the wide execution character set, used for wide string and character constants.

The default is one of UTF-32BE, UTF-32LE, UTF-16BE, or UTF-16LE, whichever corresponds to the width of "wchar_t" and the big-endian or little-endian byte order being used for code generation. As with -fexec-charset, charset can be any encoding supported by the system's "iconv" library routine; however, you will have problems with encodings that do not fit exactly in "wchar_t".

-finput-charset=charset

Set the input character set, used for translation from the character set of the input file to the source character set used by GCC. If the locale does not specify, or GCC cannot get this information from the locale, the default is UTF-8. This can be overridden by either the locale or this command-line option. Currently the command-line option takes precedence if there's a conflict. charset can be any encoding supported by the system's "iconv" library routine.

-fworking-directory

Enable generation of linemarkers in the preprocessor output that let the compiler know the current working directory at the time of preprocessing. When this option is enabled, the preprocessor emits, after the initial linemarker, a second linemarker with the current working directory followed by two slashes. GCC uses this directory, when it's present in the preprocessed input, as the directory emitted as the current working directory in some debugging information formats. This option is implicitly enabled if debugging information is enabled, but this can be inhibited with the negated form -fno-working-directory. If the -P flag is present in the command line, this option has no effect, since no "#line" directives are emitted whatsoever.

-A predicate=answer

Make an assertion with the predicate predicate and answer answer. This form is preferred to the older form -A predicate(answer), which is still supported, because it does not use shell special characters.

-A -predicate=answer

Cancel an assertion with the predicate predicate and answer answer.

-C Do not discard comments. All comments are passed through to the output file, except for comments in processed directives, which are deleted along with the directive.

You should be prepared for side effects when using -C; it causes the preprocessor to

treat comments as tokens in their own right. For example, comments appearing at the start of what would be a directive line have the effect of turning that line into an ordinary source line, since the first token on the line is no longer a #.

-CC Do not discard comments, including during macro expansion. This is like -C, except that comments contained within macros are also passed through to the output file where the macro is expanded.

In addition to the side effects of the -C option, the -CC option causes all C++-style comments inside a macro to be converted to C-style comments. This is to prevent later use of that macro from inadvertently commenting out the remainder of the source line. The -CC option is generally used to support lint comments.

-P Inhibit generation of linemarkers in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code, and will be sent to a program which might be confused by the linemarkers.

-traditional

-traditional-cpp

Try to imitate the behavior of pre-standard C preprocessors, as opposed to ISO C preprocessors.

Note that GCC does not otherwise attempt to emulate a pre-standard C compiler, and these options are only supported with the -E switch, or when invoking CPP explicitly.

-trigraphs

Support ISO C trigraphs. These are three-character sequences, all starting with ??, that are defined by ISO C to stand for single characters. For example, ??/ stands for \, so '??/n' is a character constant for a newline.

By default, GCC ignores trigraphs, but in standard-conforming modes it converts them. See the -std and -ansi options.

-remap

Enable special code to work around file systems which only permit very short file names, such as MS-DOS.

-H Print the name of each header file used, in addition to other normal activities. Each name is indented to show how deep in the #include stack it is. Precompiled header files are also printed, even if they are found to be invalid; an invalid precompiled header file is printed with ...x and a valid one with ...! .

-dletters

Says to make debugging dumps during compilation as specified by letters. The flags documented here are those relevant to the preprocessor. Other letters are interpreted by the compiler proper, or reserved for future versions of GCC, and so are silently ignored. If you specify letters whose behavior conflicts, the result is undefined.

-dM Instead of the normal output, generate a list of #define directives for all the macros defined during the execution of the preprocessor, including predefined macros. This gives you a way of finding out what is predefined in your version of the preprocessor. Assuming you have no file foo.h, the command

```
touch foo.h; cpp -dM foo.h
```

shows all the predefined macros.

-dD Like -dM except in two respects: it does not include the predefined macros, and it outputs both the #define directives and the result of preprocessing. Both kinds of output go to the standard output file.

-dN Like -dD, but emit only the macro names, not their expansions.

-dI Output #include directives in addition to the result of preprocessing.

-dU Like -dD except that only macros that are expanded, or whose definedness is tested in preprocessor directives, are output; the output is delayed until the use or test of the macro; and #undef directives are also output for macros tested but undefined at the time.

-fdebug-cpp

This option is only useful for debugging GCC. When used from CPP or with -E, it dumps debugging information about location maps. Every token in the output is preceded by the dump of the map its location belongs to.

When used from GCC without -E, this option has no effect.

-I dir

-iquote dir

-isystem dir

-idirafter dir

Add the directory dir to the list of directories to be searched for header files during preprocessing.

If dir begins with = or $SYSROOT, then the = or $SYSROOT is replaced by the sysroot prefix; see --sysroot and -isysroot.

Directories specified with -iquote apply only to the quote form of the directive,

"#include "file"". Directories specified with -I, -isystem, or -idirafter apply to lookup for both the "#include "file"" and "#include <file>" directives.

You can specify any number or combination of these options on the command line to search for header files in several directories. The lookup order is as follows:

1. For the quote form of the include directive, the directory of the current file is searched first.
2. For the quote form of the include directive, the directories specified by -iquote options are searched in left-to-right order, as they appear on the command line.
3. Directories specified with -I options are scanned in left-to-right order.
4. Directories specified with -isystem options are scanned in left-to-right order.
5. Standard system directories are scanned.
6. Directories specified with -idirafter options are scanned in left-to-right order.

You can use -I to override a system header file, substituting your own version, since these directories are searched before the standard system header file directories. However, you should not use this option to add directories that contain vendor-supplied system header files; use -isystem for that.

The -isystem and -idirafter options also mark the directory as a system directory, so that it gets the same special treatment that is applied to the standard system directories.

If a standard system include directory, or a directory specified with -isystem, is also specified with -I, the -I option is ignored. The directory is still searched but as a system directory at its normal position in the system include chain. This is to ensure that GCC's procedure to fix buggy system headers and the ordering for the "#include_next" directive are not inadvertently changed. If you really need to change the search order for system directories, use the -nostdinc and/or -isystem options.

-I- Split the include path. This option has been deprecated. Please use -iquote instead for -I directories before the -I- and remove the -I- option.

Any directories specified with -I options before -I- are searched only for headers requested with "#include "file""; they are not searched for "#include <file>". If additional directories are specified with -I options after the -I-, those directories are searched for all #include directives.

In addition, -I- inhibits the use of the directory of the current file directory as the first search directory for "#include "file"". There is no way to override this

effect of -I-.

-iprefix prefix

Specify prefix as the prefix for subsequent -iwithprefix options. If the prefix represents a directory, you should include the final /.

-iwithprefix dir

-iwithprefixbefore dir

Append dir to the prefix specified previously with -iprefix, and add the resulting directory to the include search path. -iwithprefixbefore puts it in the same place -I would; -iwithprefix puts it where -idirafter would.

-isysroot dir

This option is like the --sysroot option, but applies only to header files (except for Darwin targets, where it applies to both header files and libraries). See the --sysroot option for more information.

-imultilib dir

Use dir as a subdirectory of the directory containing target-specific C++ headers.

-nostdinc

Do not search the standard system directories for header files. Only the directories explicitly specified with -I, -iquote, -isystem, and/or -idirafter options (and the directory of the current file, if appropriate) are searched.

-nostdinc++

Do not search for header files in the C++-specific standard directories, but do still search the other standard directories. (This option is used when building the C++ library.)

-Wcomment

-Wcomments

Warn whenever a comment-start sequence /* appears in a /* comment, or whenever a backslash-newline appears in a // comment. This warning is enabled by -Wall.

-Wtrigraphs

Warn if any trigraphs are encountered that might change the meaning of the program. Trigraphs within comments are not warned about, except those that would form escaped newlines.

This option is implied by -Wall. If -Wall is not given, this option is still enabled unless trigraphs are enabled. To get trigraph conversion without warnings, but get

the other -Wall warnings, use -trigraphs -Wall -Wno-trigraphs.

-Wundef

Warn if an undefined identifier is evaluated in an "#if" directive. Such identifiers are replaced with zero.

-Wexpansion-to-defined

Warn whenever defined is encountered in the expansion of a macro (including the case where the macro is expanded by an #if directive). Such usage is not portable. This warning is also enabled by -Wpedantic and -Wextra.

-Wunused-macros

Warn about macros defined in the main file that are unused. A macro is used if it is expanded or tested for existence at least once. The preprocessor also warns if the macro has not been used at the time it is redefined or undefined.

Built-in macros, macros defined on the command line, and macros defined in include files are not warned about.

Note: If a macro is actually used, but only used in skipped conditional blocks, then the preprocessor reports it as unused. To avoid the warning in such a case, you might improve the scope of the macro's definition by, for example, moving it into the first skipped block. Alternatively, you could provide a dummy use with something like:

```
#if defined the_macro_causing_the_warning
#endif
```

-Wno-endif-labels

Do not warn whenever an "#else" or an "#endif" are followed by text. This sometimes happens in older programs with code of the form

```
#if FOO
...
#else FOO
...
#endif FOO
```

The second and third "FOO" should be in comments. This warning is on by default.

ENVIRONMENT

This section describes the environment variables that affect how CPP operates. You can use them to specify directories or prefixes to use when searching for include files, or to control dependency output.

Note that you can also specify places to search using options such as -I, and control dependency output with options like -M. These take precedence over environment variables, which in turn take precedence over the configuration of GCC.

CPATH

C_INCLUDE_PATH

CPLUS_INCLUDE_PATH

OBJC_INCLUDE_PATH

Each variable's value is a list of directories separated by a special character, much like PATH, in which to look for header files. The special character, "PATH_SEPARATOR", is target-dependent and determined at GCC build time. For Microsoft Windows-based targets it is a semicolon, and for almost all other targets it is a colon.

CPATH specifies a list of directories to be searched as if specified with -I, but after any paths given with -I options on the command line. This environment variable is used regardless of which language is being preprocessed.

The remaining environment variables apply only when preprocessing the particular language indicated. Each specifies a list of directories to be searched as if specified with -isystem, but after any paths given with -isystem options on the command line.

In all these variables, an empty element instructs the compiler to search its current working directory. Empty elements can appear at the beginning or end of a path. For instance, if the value of CPATH is ":/special/include", that has the same effect as -I. -I/special/include.

DEPENDENCIES_OUTPUT

If this variable is set, its value specifies how to output dependencies for Make based on the non-system header files processed by the compiler. System header files are ignored in the dependency output.

The value of DEPENDENCIES_OUTPUT can be just a file name, in which case the Make rules are written to that file, guessing the target name from the source file name. Or the value can have the form file target, in which case the rules are written to file file using target as the target name.

In other words, this environment variable is equivalent to combining the options -MM and -MF, with an optional -MT switch too.

SUNPRO_DEPENDENCIES

This variable is the same as DEPENDENCIES_OUTPUT (see above), except that system header files are not ignored, so it implies -M rather than -MM. However, the dependence on the main input file is omitted.

SOURCE_DATE_EPOCH

If this variable is set, its value specifies a UNIX timestamp to be used in replacement of the current date and time in the "__DATE__" and "__TIME__" macros, so that the embedded timestamps become reproducible.

The value of SOURCE_DATE_EPOCH must be a UNIX timestamp, defined as the number of seconds (excluding leap seconds) since 01 Jan 1970 00:00:00 represented in ASCII; identical to the output of "date +%s" on GNU/Linux and other systems that support the %s extension in the "date" command.

The value should be a known timestamp such as the last modification time of the source or package and it should be set by the build process.

## SEE ALSO

gpl(7), gfdl(7), fsf-funding(7), gcc(1), and the Info entries for cpp and gcc.

## COPYRIGHT