Next: Assembler Options, Previous: Optimize Options, Up: Invoking GCC

3.11 Options Controlling the Preprocessor

These options control the C preprocessor, which is run on each C source file before actual compilation.

If you use the `-E` option, nothing is done except preprocessing. Some of these options make sense only together with `-E` because they cause the preprocessor output to be unsuitable for actual compilation.

You can use `-Wp,`*`option`* to bypass the compiler driver and pass *option* directly through to the preprocessor. If *option* contains commas, it is split into multiple options at the commas. However, many options are modified, translated or interpreted by the compiler driver before being passed to the preprocessor, and `-Wp` forcibly bypasses this phase. The preprocessor's direct interface is undocumented and subject to change, so whenever possible you should avoid using `-Wp` and let the driver handle the options instead.

`-Xpreprocessor` *option*

Pass *option* as an option to the preprocessor. You can use this to supply system-specific preprocessor options which GCC does not know how to recognize.

If you want to pass an option that takes an argument, you must use `-Xpreprocessor` twice, once for the option and once for the argument.

`-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 will be 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 will need to 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.

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

`-I` *dir*
: Add the directory *dir* to the list of directories to be searched for header files. Directories named by `-I` are searched before the standard system include directories. If the directory *dir* is a standard system include directory, the option is ignored to ensure that the default search order for system directories and the special treatment of system headers are not defeated . If *dir* begins with `=`, then the `=` will be replaced by the sysroot prefix; see `--sysroot` and `-isysroot`.

`-o` *file*
: Write output to *file*. This is the same as specifying *file* as the second non-option argument to `cpp`. `gcc` has a different interpretation of a second non-option argument, so you must use `-o` to specify the output file.

`-Wall`
: Turns on all optional warnings which are desirable for normal code. At present this is `-Wcomment`, `-Wtrigraphs`, `-Wmultichar` and a warning about integer promotion causing a change of sign in `#if` expressions. Note that many of the preprocessor's warnings are on by default and have no options to control them.

`-Wcomment`

`-Wcomments`
: Warn whenever a comment-start sequence `/*' appears in a `/*' comment, or whenever a backslash-newline appears in a `//' comment. (Both forms have the same effect.)

`-Wtrigraphs`
: Most trigraphs in comments cannot affect the meaning of the program. However, a trigraph that would form an escaped newline (`??/' at the end of a line) can, by changing where the comment begins or ends. Therefore, only trigraphs that would form escaped newlines produce warnings inside a comment.

  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'.

`-Wtraditional`
: Warn about certain constructs that behave differently in traditional and ISO C. Also warn about ISO C constructs that have no traditional C equivalent, and problematic constructs which should be avoided.

`-Wundef`
: Warn whenever an identifier which is not a macro is encountered in an `#if'

directive, outside of `defined'. Such identifiers are replaced with zero.

`-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 will also warn 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 CPP will report 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
```

`-Wendif-labels`
: Warn whenever an `#else' or an `#endif' are followed by text. This usually happens in code of the form

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

  The second and third `FOO` should be in comments, but often are not in older programs. This warning is on by default.

`-Werror`
: Make all warnings into hard errors. Source code which triggers warnings will be rejected.

`-Wsystem-headers`
: Issue warnings for code in system headers. These are normally unhelpful in finding bugs in your own code, therefore suppressed. If you are responsible for the system library, you may want to see them.

`-w`
: Suppress all warnings, including those which GNU CPP issues by default.

`-pedantic`
: Issue all the mandatory diagnostics listed in the C standard. Some of them are left out by default, since they trigger frequently on harmless code.

`-pedantic-errors`
: Issue all the mandatory diagnostics, and make all mandatory diagnostics into errors. This includes mandatory diagnostics that GCC issues without `-pedantic' but treats as warnings.

`-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` (see Environment Variables). Debug output will still be 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 will appear in `-MM` dependency output. This is a slight change in semantics from GCC versions 3.0 and earlier.

`-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 have sent preprocessed output.

When used with the driver options `-MD` or `-MMD`, `-MF` overrides the default dependency output file.

`-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.

`-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 will set 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 (see -MF), 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.

`-fpch-deps`
When using precompiled headers (see Precompiled Headers), this flag will cause the dependency-output flags to also list the files from the precompiled header's dependencies. If not specified only the precompiled header would be listed and not the files that were used to create it because those files are not consulted when a precompiled header is used.

`-fpch-preprocess`
This option allows use of a precompiled header (see Precompiled Headers) together with `-E`. It inserts a special `#pragma`, `#pragma GCC pch_preprocess "<filename>"` in the output to mark the place where the precompiled header was found, and its filename. When `-fpreprocessed` is in use, GCC recognizes this `#pragma` and loads the PCH.

This option is off by default, because the resulting preprocessed output is only really suitable as input to GCC. It is switched on by `-save-temps`.

You should not write this `#pragma` in your own code, but it is safe to edit the filename if the PCH file is available in a different location. The filename may be absolute or it may be relative to GCC's current directory.

`-x c`
`-x c++`
`-x objective-c`
`-x assembler-with-cpp`
Specify the source language: C, C++, Objective-C, or assembly. This has nothing to do with standards conformance or extensions; it merely selects which base syntax to expect. If you give none of these options, cpp will deduce the language from the extension of the source file: `.c', `.cc', `.m', or `.S'. Some other common extensions for C++ and assembly are also recognized. If cpp does not recognize the extension, it will treat the file as C; this is the most generic mode.

Note: Previous versions of cpp accepted a `-lang` option which selected both the language and the standards conformance level. This option has been removed, because it conflicts with the `-l` option.

`-std=`*standard*
`-ansi`
Specify the standard to which the code should conform. Currently CPP knows about C and C++ standards; others may be added in the future.

standard may be one of:

`iso9899:1990`
`c89`
: The ISO C standard from 1990. `c89' is the customary shorthand for this version of the standard.

  The `-ansi` option is equivalent to `-std=c89`.

`iso9899:199409`
: The 1990 C standard, as amended in 1994.

`iso9899:1999`
`c99`
`iso9899:199x`
`c9x`
: The revised ISO C standard, published in December 1999. Before publication, this was known as C9X.

`gnu89`
: The 1990 C standard plus GNU extensions. This is the default.

`gnu99`
`gnu9x`
: The 1999 C standard plus GNU extensions.

`c++98`
: The 1998 ISO C++ standard plus amendments.

`gnu++98`
: The same as `-std=c++98` plus GNU extensions. This is the default for C++ code.

`-I-`
: Split the include path. 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*`"`. This option has been deprecated.

`-nostdinc`
: Do not search the standard system directories for header files. Only the directories you have specified with `-I` 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.)

`-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`.

`-idirafter` *dir*
: Search *dir* for header files, but do it *after* all directories specified with `-I` and the standard system directories have been exhausted. *dir* is treated as a system include directory. If *dir* begins with `=`, then the `=` will be replaced by the sysroot prefix; see `--sysroot` and `-isysroot`.

`-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. See the `--sysroot` option for more information.

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

`-isystem` *dir*
: Search *dir* for header files, after all directories specified by `-I` but before the standard system directories. Mark it as a system directory, so that it gets the same special treatment as is applied to the standard system directories. If *dir* begins with `=`, then the `=` will be replaced by the sysroot prefix; see `--sysroot` and `-isysroot`.

`-iquote` *dir*
: Search *dir* only for header files requested with `#include "`*file*`"`; they are not searched for `#include <`*file*`>`, before all directories specified by `-I` and before the standard system directories. If *dir* begins with `=`, then the `=` will be

replaced by the sysroot prefix; see `--sysroot` and `-isysroot`.

`-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 in identifiers. This option is experimental; in a future version of GCC, it will be enabled by default for C99 and C++.

`-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`.

`-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.

`-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 UTF-32 or UTF-16, whichever corresponds to the width of `wchar_t`. 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 will let the compiler know the current working directory at the time of preprocessing. When this option is enabled, the preprocessor will emit, after the initial linemarker, a second linemarker with the current working directory followed by two slashes. GCC will use 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.

`-fno-show-column`
: Do not print column numbers in diagnostics. This may be necessary if diagnostics are being scanned by a program that does not understand the column numbers, such as `dejagnu`.

`-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*.

`-dCHARS`
: *CHARS* is a sequence of one or more of the following characters, and must not be preceded by a space. Other characters are interpreted by the compiler proper, or reserved for future versions of GCC, and so are silently ignored. If you specify characters whose behavior conflicts, the result is undefined.

  `` `M' ``
  : 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
```

will show all the predefined macros.

If you use `-dM` without the `-E` option, `-dM` is interpreted as a synonym for `-fdump-rtl-mach`. See Debugging Options.

`D'
Like `M' 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.

`N'
Like `D', but emit only the macro names, not their expansions.

`I'
Output `#include' directives in addition to the result of preprocessing.

`U'
Like `D' 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.

`-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.

`-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.

`-traditional-cpp`
: Try to imitate the behavior of old-fashioned C preprocessors, as opposed to ISO C preprocessors.

`-trigraphs`
: Process trigraph sequences. 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.

  The nine trigraphs and their replacements are

```
Trigraph:       ??(  ??)  ??<  ??>  ??=  ??/  ??'  ??!  ??-
Replacement:      [    ]    {    }    #    \    ^    |    ~
```

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

`--help`
`--target-help`
: Print text describing all the command line options instead of preprocessing anything.

`-v`
: Verbose mode. Print out GNU CPP's version number at the beginning of execution, and report the final form of the include path.

`-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 `...!' .

`-version`
`--version`
: Print out GNU CPP's version number. With one dash, proceed to preprocess as normal. With two dashes, exit immediately.