Clang-Tidy

Contents

See also:

- **The list of clang-tidy checks**
- **Clang-tidy IDE/Editor Integrations**
- **Getting Involved**

clang-tidy is a clang-based C++ “linter” tool. Its purpose is to provide an extensible framework for diagnosing and fixing typical programming errors, like style violations, interface misuse, or bugs that can be deduced via static analysis. **clang-tidy** is modular and provides a convenient interface for writing new checks.

Using clang-tidy

clang-tidy is a **LibTooling**-based tool, and it’s easier to work with if you set up a compile command database for your project (for an example of how to do this, see **How To Setup Tooling For LLVM**). You can also specify compilation options on the command line after `--`:

```
$ clang-tidy test.cpp -- -Imy_project/include -DMY_DEFINES ...
```

clang-tidy has its own checks and can also run Clang Static Analyzer checks. Each check has a name and the checks to run can be chosen using the `-checks=` option, which specifies a comma-separated list of positive and negative (prefixed with `-`) globs. Positive globs add subsets of checks, and negative globs remove them. For example,

```
$ clang-tidy test.cpp -checks=-*,clang-analyzer-*,-clang-analyzer-cplusplus*
```

will disable all default checks (`-*`) and enable all `clang-analyzer-*` checks except for `clang-analyzer-cplusplus*` ones.

The `-list-checks` option lists all the enabled checks. When used without `-checks=`, it shows checks enabled by default. Use `-checks=*` to see all available checks or with any other value of `-checks=` to see which checks are enabled by this value.

There are currently the following groups of checks:

Name prefix	Description
`abseil-`	Checks related to Abseil library.
`altera-`	Checks related to OpenCL programming for FPGAs.
`android-`	Checks related to Android.
`boost-`	Checks related to Boost library.
`bugprone-`	Checks that target bug-prone code constructs.
`cert-`	Checks related to CERT Secure Coding Guidelines.
`clang-analyzer-`	Clang Static Analyzer checks.
`concurrency-`	Checks related to concurrent programming (including threads, fibers, coroutines, etc.).
`cppcoreguidelines-`	Checks related to C++ Core Guidelines.
`darwin-`	Checks related to Darwin coding conventions.
`fuchsia-`	Checks related to Fuchsia coding conventions.
`google-`	Checks related to Google coding conventions.
`hicpp-`	Checks related to High Integrity C++ Coding Standard.
`linuxkernel-`	Checks related to the Linux Kernel coding conventions.
`llvm-`	Checks related to the LLVM coding conventions.
`llvmlibc-`	Checks related to the LLVM-libc coding standards.
`misc-`	Checks that we didn’t have a better category for.
`modernize-`	Checks that advocate usage of modern (currently “modern” means “C++11”) language constructs.
`mpi-`	Checks related to MPI (Message Passing Interface).
`objc-`	Checks related to Objective-C coding conventions.

Name prefix	Description
openmp-	Checks related to OpenMP API.
performance-	Checks that target performance-related issues.
portability-	Checks that target portability-related issues that don't relate to any particular coding style.
readability-	Checks that target readability-related issues that don't relate to any particular coding style.
zircon-	Checks related to Zircon kernel coding conventions.

Clang diagnostics are treated in a similar way as check diagnostics. Clang diagnostics are displayed by **clang-tidy** and can be filtered out using the `-checks=` option. However, the `-checks=` option does not affect compilation arguments, so it cannot turn on Clang warnings which are not already turned on in the build configuration. The `-warnings-as-errors=` option upgrades any warnings emitted under the `-checks=` flag to errors (but it does not enable any checks itself).

Clang diagnostics have check names starting with `clang-diagnostic-`. Diagnostics which have a corresponding warning option, are named `clang-diagnostic-<warning-option>`, e.g. Clang warning controlled by `-Wliteral-conversion` will be reported with check name `clang-diagnostic-literal-conversion`.

The `-fix` flag instructs **clang-tidy** to fix found errors if supported by corresponding checks.

An overview of all the command-line options:

```
$ clang-tidy --help
USAGE: clang-tidy [options] <source0> [... <sourceN>]

OPTIONS:

Generic Options:

  --help                      - Display available options (--help-hidden for more)
  --help-list                 - Display list of available options (--help-list-hidden for more)
  --version                   - Display the version of this program

clang-tidy options:

  --checks=<string>           -
                                Comma-separated list of globs with optional '-'
                                prefix. Globs are processed in order of
                                appearance in the list. Globs without '-'
                                prefix add checks with matching names to the
                                set, globs with the '-' prefix remove checks
                                with matching names from the set of enabled
                                checks. This option's value is appended to the
                                value of the 'Checks' option in .clang-tidy
                                file, if any.
  --config=<string>           -
                                Specifies a configuration in YAML/JSON format:
                                  -config="{Checks: '*',
                                            CheckOptions: [{key: x,
                                                            value: y}]}"
                                When the value is empty, clang-tidy will
                                attempt to find a file named .clang-tidy for
                                each source file in its parent directories.
  --config-file=<string>      -
                                Specify the path of .clang-tidy or custom config file:
                                  e.g. --config-file=/some/path/myTidyConfigFile
                                This option internally works exactly the same way as
                                  --config option after reading specified config file.
                                Use either --config-file or --config, not both.
  --dump-config               -
                                Dumps configuration in the YAML format to
                                stdout. This option can be used along with a
                                file name (and '--' if the file is outside of a
                                project with configured compilation database).
                                The configuration used for this file will be
                                printed.
                                Use along with -checks=* to include
                                configuration of all checks.
  --enable-check-profile      -
                                Enable per-check timing profiles, and print a
                                report to stderr.
  --explain-config            -
                                For each enabled check explains, where it is
                                enabled, i.e. in clang-tidy binary, command
                                line or a specific configuration file.
  --export-fixes=<filename>   -
                                YAML file to store suggested fixes in. The
                                stored fixes can be applied to the input source
                                code with clang-apply-replacements.
  --extra-arg=<string>        - Additional argument to append to the compiler command line.
                                Can be used several times.
  --extra-arg-before=<string> - Additional argument to prepend to the compiler command line.
                                Can be used several times.
  --fix                       -
                                Apply suggested fixes. Without -fix-errors
                                clang-tidy will bail out if any compilation
```

```
                                errors were found.
  --fix-errors                  -
                                Apply suggested fixes even if compilation
                                errors were found. If compiler errors have
                                attached fix-its, clang-tidy will apply them as
                                well.
  --fix-notes                   -
                                If a warning has no fix, but a single fix can
                                be found through an associated diagnostic note,
                                apply the fix.
                                Specifying this flag will implicitly enable the
                                '--fix' flag.
  --format-style=<string>       -
                                Style for formatting code around applied fixes:
                                  - 'none' (default) turns off formatting
                                  - 'file' (literally 'file', not a placeholder)
                                    uses .clang-format file in the closest parent
                                    directory
                                  - '{ <json> }' specifies options inline, e.g.
                                    -format-style='{BasedOnStyle: llvm, IndentWidth: 8}'
                                  - 'llvm', 'google', 'webkit', 'mozilla'
                                See clang-format documentation for the up-to-date
                                information about formatting styles and options.
                                This option overrides the 'FormatStyle` option in
                                .clang-tidy file, if any.
  --header-filter=<string>      -
                                Regular expression matching the names of the
                                headers to output diagnostics from. Diagnostics
                                from the main file of each translation unit are
                                always displayed.
                                Can be used together with -line-filter.
                                This option overrides the 'HeaderFilterRegex'
                                option in .clang-tidy file, if any.
  --line-filter=<string>        -
                                List of files with line ranges to filter the
                                warnings. Can be used together with
                                -header-filter. The format of the list is a
                                JSON array of objects:
                                  [
                                    {"name":"file1.cpp","lines":[[1,3],[5,7]]},
                                    {"name":"file2.h"}
                                  ]
  --list-checks                 -
                                List all enabled checks and exit. Use with
                                -checks=* to list all available checks.
  -load=<plugin>                -
                                Load the dynamic object ``plugin``. This
                                object should register new static analyzer
                                or clang-tidy passes. Once loaded, the
                                object will add new command line options
                                to run various analyses. To see the new
                                complete list of passes, use the
                                :option:`--list-checks` and
                                :option:`-load` options together.
  -p=<string>                   - Build path
  --quiet                       -
                                Run clang-tidy in quiet mode. This suppresses
                                printing statistics about ignored warnings and
                                warnings treated as errors if the respective
                                options are specified.
  --store-check-profile=<prefix> -
                                By default reports are printed in tabulated
                                format to stderr. When this option is passed,
                                these per-TU profiles are instead stored as JSON.
  --system-headers              - Display the errors from system headers.
  --use-color                   -
                                Use colors in diagnostics. If not set, colors
                                will be used if the terminal connected to
                                standard output supports colors.
                                This option overrides the 'UseColor' option in
                                .clang-tidy file, if any.
  --vfsoverlay=<filename>       -
                                Overlay the virtual filesystem described by file
                                over the real file system.
  --warnings-as-errors=<string>  -
                                Upgrades warnings to errors. Same format as
                                '-checks'.
                                This option's value is appended to the value of
                                the 'WarningsAsErrors' option in .clang-tidy
                                file, if any.

-p <build-path> is used to read a compile command database.

        For example, it can be a CMake build directory in which a file named
        compile_commands.json exists (use -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
        CMake option to get this output). When no build path is specified,
        a search for compile_commands.json will be attempted through all
        parent paths of the first input file . See:
        https://clang.llvm.org/docs/HowToSetupToolingForLLVM.html for an
        example of setting up Clang Tooling on a source tree.
```

```
<source0> ... specify the paths of source files. These paths are
        looked up in the compile command database. If the path of a file is
        absolute, it needs to point into CMake's source tree. If the path is
        relative, the current working directory needs to be in the CMake
        source tree and the file must be in a subdirectory of the current
        working directory. "./" prefixes in the relative files will be
        automatically removed, but the rest of a relative path must be a
        suffix of a path in the compile command database.


Configuration files:
  clang-tidy attempts to read configuration for each source file from a
  .clang-tidy file located in the closest parent directory of the source
  file. If InheritParentConfig is true in a config file, the configuration file
  in the parent directory (if any exists) will be taken and current config file
  will be applied on top of the parent one. If any configuration options have
  a corresponding command-line option, command-line option takes precedence.
  The effective configuration can be inspected using -dump-config:

    $ clang-tidy -dump-config
    ---
    Checks:              '-*,some-check'
    WarningsAsErrors:    ''
    HeaderFilterRegex:   ''
    FormatStyle:         none
    InheritParentConfig: true
    User:                user
    CheckOptions:
      - key:             some-check.SomeOption
        value:           'some value'
    ...
```

Suppressing Undesired Diagnostics

clang-tidy diagnostics are intended to call out code that does not adhere to a coding standard, or is otherwise problematic in some way. However, if the code is known to be correct, it may be useful to silence the warning. Some clang-tidy checks provide a check-specific way to silence the diagnostics, e.g. **bugprone-use-after-move** can be silenced by re-initializing the variable after it has been moved out, **bugprone-string-integer-assignment** can be suppressed by explicitly casting the integer to `char`, **readability-implicit-bool-conversion** can also be suppressed by using explicit casts, etc.

If a specific suppression mechanism is not available for a certain warning, or its use is not desired for some reason, **clang-tidy** has a generic mechanism to suppress diagnostics using `NOLINT`, `NOLINTNEXTLINE`, and `NOLINTBEGIN` … `NOLINTEND` comments.

The `NOLINT` comment instructs **clang-tidy** to ignore warnings on the *same line* (it doesn't apply to a function, a block of code or any other language construct; it applies to the line of code it is on). If introducing the comment on the same line would change the formatting in an undesired way, the `NOLINTNEXTLINE` comment allows suppressing clang-tidy warnings on the *next line*. The `NOLINTBEGIN` and `NOLINTEND` comments allow suppressing clang-tidy warnings on *multiple lines* (affecting all lines between the two comments).

All comments can be followed by an optional list of check names in parentheses (see below for the formal syntax). The list of check names supports globbing, with the same format and semantics as for enabling checks. Note: negative globs are ignored here, as they would effectively re-activate the warning.

For example:

```
class Foo {
  // Suppress all the diagnostics for the line
  Foo(int param); // NOLINT

  // Consider explaining the motivation to suppress the warning
  Foo(char param); // NOLINT: Allow implicit conversion from `char`, because <some valid reason>

  // Silence only the specified checks for the line
  Foo(double param); // NOLINT(google-explicit-constructor, google-runtime-int)

  // Silence all checks from the `google` module
  Foo(bool param); // NOLINT(google*)

  // Silence all checks ending with `-avoid-c-arrays`
  int array[10]; // NOLINT(*-avoid-c-arrays)

  // Silence only the specified diagnostics for the next line
  // NOLINTNEXTLINE(google-explicit-constructor, google-runtime-int)
  Foo(bool param);

  // Silence all checks from the `google` module for the next line
  // NOLINTNEXTLINE(google*)
  Foo(bool param);

  // Silence all checks ending with `-avoid-c-arrays` for the next line
  // NOLINTNEXTLINE(*-avoid-c-arrays)
  int array[10];
```

```
  // Silence only the specified checks for all lines between the BEGIN and END
  // NOLINTBEGIN(google-explicit-constructor, google-runtime-int)
  Foo(short param);
  Foo(long param);
  // NOLINTEND(google-explicit-constructor, google-runtime-int)

  // Silence all checks from the `google` module for all lines between the BEGIN and END
  // NOLINTBEGIN(google*)
  Foo(bool param);
  // NOLINTEND(google*)

  // Silence all checks ending with `-avoid-c-arrays` for all lines between the BEGIN and END
  // NOLINTBEGIN(*-avoid-c-arrays)
  int array[10];
  // NOLINTEND(*-avoid-c-arrays)
};
```

The formal syntax of `NOLINT`, `NOLINTNEXTLINE`, and `NOLINTBEGIN` … `NOLINTEND` is the following:

```
lint-comment:
  lint-command
  lint-command lint-args

lint-args:
  ( check-name-list )

check-name-list:
  check-name
  check-name-list , check-name

lint-command:
  NOLINT
  NOLINTNEXTLINE
  NOLINTBEGIN
  NOLINTEND
```

Note that whitespaces between `NOLINT`/`NOLINTNEXTLINE`/`NOLINTBEGIN`/`NOLINTEND` and the opening parenthesis are not allowed (in this case the comment will be treated just as `NOLINT`/`NOLINTNEXTLINE`/`NOLINTBEGIN`/`NOLINTEND`), whereas in the check names list (inside the parentheses), whitespaces can be used and will be ignored.

All `NOLINTBEGIN` comments must be paired by an equal number of `NOLINTEND` comments. Moreover, a pair of comments must have matching arguments – for example, `NOLINTBEGIN(check-name)` can be paired with `NOLINTEND(check-name)` but not with `NOLINTEND` *(zero arguments)*. **clang-tidy** will generate a `clang-tidy-nolint` error diagnostic if any `NOLINTBEGIN`/`NOLINTEND` comment violates these requirements.