Clang tools Format, lint, analyze C/C++

START HERE

`clang-format -i <file>`
Format a file in place

`clang-format --style=llvm --dump-config > .clang-format`
Start a style file

`git clang-format`
Format only changed lines

`clang --analyze <file>`
Static analysis

`clang-tidy -p build <file>`
Lint with real build flags

`clang-tidy --fix -p build <file>`
Apply suggested fixes

`clang -Weverything <file>`
See every warning clang has

`clang -ftime-trace -c <file>`
Profile a slow compile

CMake, Meson writes → **compile_commands.json** -p build → **clang-tidy, clangd** report → **warnings + fixes**

One compile database feeds the analysis tools

CONCEPTS

Each tool reads its own YAML file from the nearest parent directory: .clang-format, .clang-tidy, .clangd.

clang-format only touches whitespace and layout; clang-tidy needs the exact compile flags to parse your code.

CLANG-FORMAT

`clang-format <file>`
Print the formatted file to stdout

`clang-format -i <a.cpp> <b.h>`
Format files in place

`clang-format --style=google -i <file>`
Use a preset style
Also llvm, gnu, chromium, microsoft, mozilla, webkit.

`clang-format --dry-run --Werror <file>`
Fail if a file is not formatted

`clang-format --lines=<start>:<end> -i <file>`
Format a line range only

STYLE FILES

`clang-format --style=llvm --dump-config`
Print every option of a preset

`clang-format --dump-config <file>`
Show the style used for a file

.clang-format

```
BasedOnStyle: LLVM
IndentWidth: 4
ColumnLimit: 100
---
Language: Proto
DisableFormat: true
```

`// clang-format off`
Stop formatting until ... on
Closed by // clang-format on. / clang-format off */ works too.*

`.clang-format-ignore`
Glob list of files to skip

FORMAT ONLY CHANGES

`git clang-format`
Format lines changed vs HEAD

`git clang-format <branch>`
Format all changes since <branch>

`git clang-format --diff`
Print the diff, change nothing

FORMAT ONLY CHANGES (cont.)

`git clang-format --staged`
Format lines in the index

CLANG-TIDY

`clang-tidy -p <build> <file>`
Lint with flags from compile_commands

`clang-tidy <file> -- <compiler flags>`
Lint without a compile database

`clang-tidy --checks='-*,bugprone-*' <file>`
Pick checks by glob
Later globs win; a leading - disables.

`clang-tidy --fix -p <build> <file>`
Apply the fixes

`clang-tidy --list-checks --checks='*'`
List every available check

`clang-tidy --dump-config`
Print the effective config

`clang-tidy --header-filter='<regex>' <file>`
Also report in matching headers

`clang-tidy --warnings-as-errors='*' <file>`
Fail CI on any finding

`run-clang-tidy -p <build> -j 8`
Lint the whole database in parallel
Also named run-clang-tidy.py.

CHECK GROUPS

Prefix	Looks for
`bugprone-`	Likely bugs
`modernize-`	Newer idioms
`performance-`	Wasted copies
`readability-`	Clarity
`cppcoreguidelines-`	Guidelines
`clang-analyzer-`	Analyzer

CONFIGURE AND SILENCE TIDY

.clang-tidy

```
Checks: '-*,bugprone-*,performance-*'
WarningsAsErrors: 'bugprone-*'
HeaderFilterRegex: 'src/.*'
CheckOptions:
  <check>.<Option>: <value>
```

`// NOLINT`
Silence findings on this line

CONFIGURE AND SILENCE TIDY (cont.)

`// NOLINT(<check>)`
Silence one check on this line
No space before the parenthesis, or it acts as a bare NOLINT.

`// NOLINTNEXTLINE(<check>)`
Silence the next line

`// NOLINTBEGIN(<check>)`
Silence until NOLINTEND(<check>)

CLANGD

clangd is the C/C++ language server most editors use. It finds compile_commands.json in a parent dir or its build/ subdir.

.clangd

```
CompileFlags:
  CompilationDatabase: build
  Add: [-Wall]
  Remove: [-fno-*]
Diagnostics:
  UnusedIncludes: Strict
  ClangTidy:
    Add: [bugprone-*]
```

`clangd --check=<file>`
Debug how clangd sees one file

CLANG-ONLY WARNINGS

`clang -Weverything <file>`
Every warning clang knows
Noisy and partly contradictory: for discovery only.

`clang -fsyntax-only <file>`
Check code, produce no output

`clang -fcolor-diagnostics <file>`
Force color (e.g. through ninja)

Silence a warning in one region

```
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunused-parameter"
/* ... */
#pragma clang diagnostic pop
```

SANITIZERS AND COVERAGE

`clang -fsanitize=memory -g <file>`
MemorySanitizer: uninitialized reads
Linux only; every linked library must be instrumented too.

`clang -fprofile-instr-generate -fcoverage-mapping <file>`
Source-based coverage build

SANITIZERS AND COVERAGE (cont.)

`llvm-cov report <app> -instr-profile=<p.profdata>`
Coverage summary per file

STATIC ANALYZER

`clang --analyze <file>`
Analyze one file

`clang --analyze --analyzer-output text <file>`
Print findings to the terminal

`scan-build make`
Analyze a whole build

`scan-build -o <dir> -V make`
HTML report, open in browser

COMPILE-TIME INSIGHT

`clang -ftime-trace -c <file>`
Write a Chrome trace of the compile
Open the .json in chrome://tracing, Perfetto or speedscope.

`clang -O2 -Rpass=inline <file>`
Report what got inlined

`clang -S -emit-llvm <file>`
Write LLVM IR (.ll)

TOOLCHAIN CHOICES

`clang -fuse-ld=lld <file>`
Link with LLVM's fast lld

`CC=clang CXX=clang++ cmake -B build`
Build a CMake project with clang

RECIPES

Format check in CI

```
git ls-files '*.c' '*.h' '*.cpp' '*.hpp' \
  | xargs clang-format --dry-run --Werror
```

Lint a CMake project

```
export CMAKE_EXPORT_COMPILE_COMMANDS=1
cmake -B build
run-clang-tidy -p build -j 8 -quiet
# apply fixes
run-clang-tidy -p build -fix -checks='-*,modernize-*'
```

Find the slowest headers

```
CXX=clang++ CXXFLAGS=-ftime-trace cmake -B build
cmake --build build
# open build/**/*.json in ui.perfetto.dev
```