Cppcheck C/C++ static analyser

START HERE

`cppcheck <path>`
Check files or a folder

`cppcheck -j 4 <path>`
Use 4 threads

`cppcheck --enable=all <path>`
Turn on more checks

`cppcheck --std=c++17 <path>`
Pick the language standard

`cppcheck -I <dir> <path>`
Add an include path

`cppcheck --suppress=<id> <path>`
Silence one message

`cppcheck --project=compile_commands.json`
Check what you build

`cppcheck --error-exitcode=1 <path>`
Fail the build on findings

Sources files or project → `preprocess` → **Configurations** #ifdef variants → `analyse` → **Checks** --enable ids → `report` → **Findings** text / XML

CONCEPTS

By default only errors are shown. Add --enable=style (or all) to see warnings, style and performance findings.

Cppcheck tries each #ifdef configuration itself, so missing -I paths and -D flags give fewer findings, not errors. Use --check-config to see what it could not resolve.

RUNNING

`cppcheck <file>`
Check one file

`cppcheck <dir>`
Check all sources in a folder, recursively

`cppcheck -j <n> <dir>`
Check with n threads

`cppcheck -q <dir>`
Quiet: print only findings

`cppcheck -i <path> <dir>`
Skip a file or folder
Example: -isrc/third_party. Matches by path prefix or pattern.

`cppcheck <dir> --file-filter=<pattern>`
Check only matching files

`cppcheck --language=c++ <file>`
Force C or C++ for all files

`cppcheck --showtime=summary <dir>`
Show where the time goes

CHECKS AND LEVELS

`cppcheck --enable=warning <dir>`
Add warnings

`cppcheck --enable=style <dir>`
Add style findings

`cppcheck --enable=performance,portability <dir>`
Add several groups (comma list)

`cppcheck --enable=all <dir>`
Enable every group
all includes unusedFunction, which gives false positives unless you pass the whole program.

`cppcheck --enable=unusedFunction <dir>`
Find unused functions

CHECKS AND LEVELS (cont.)

`cppcheck --enable=missingInclude <dir>`
Report unresolved includes

`cppcheck --check-level=exhaustive <dir>`
Deeper, slower analysis

`cppcheck --inconclusive <dir>`
Also report uncertain findings

`cppcheck --check-config <dir>`
Show missing includes and config problems

CONFIGURATION

`cppcheck -I <dir> <path>`
Add an include directory

`cppcheck -D<name> <path>`
Define a macro, check only that config

`cppcheck -U<name> <path>`
Treat a macro as undefined

`cppcheck --max-configs=<n> <path>`
Limit #ifdef configurations tried

`cppcheck --force <path>`
Check every configuration (can be slow)

`cppcheck --std=c++20 <path>`
Set the C++ standard
C++: c++03 to c++26. C: c89, c99, c11, c17, c23.

`cppcheck --std=c11 <file>`
Set the C standard

`cppcheck --platform=<name> <path>`
Set target types and sizes
Builtin names include unix32, unix64, win32A, win32W, win64. Custom platforms use an XML file.

`cppcheck --library=<name> <path>`
Load a library config (posix, qt, ...)

PROJECTS AND INCREMENTAL RUNS

`cppcheck --project=compile_commands.json`
Use a compilation database

`cppcheck --project=<file>.sln`
Import a Visual Studio solution

PROJECTS AND INCREMENTAL RUNS (cont.)

`cppcheck --project=<file>.vcxproj`
Import a Visual Studio project

`cppcheck --project=<file>.cppcheck`
Use a GUI project file

`cppcheck --project=compile_commands.json -i<dir>`
Exclude a folder from a project

`cppcheck --cppcheck-build-dir=<dir> <path>`
Cache results, enable whole-program checks
Reuse the same folder between runs for incremental analysis. Needed for cross-file checks with -j.

OUTPUT

`cppcheck --template=gcc <path>`
GCC-style lines for editors

`cppcheck --template=vs <path>`
Visual Studio-style lines

`cppcheck --template='{file}:{line}: {message}' <path>`
Custom line format
Other fields: {callstack}, {code}, {inconclusive:text}, {remark}. Use \n and \t for breaks.

`cppcheck --xml <path> 2> <out.xml>`
XML report (written to stderr)

`cppcheck-htmlreport --file=<out.xml> --report-dir=<dir> --source-dir=.`
Turn XML into an HTML report
Needs Python and pygments.

`cppcheck -rp <path>`
Print paths relative to a base

`cppcheck --error-exitcode=<n>`
Exit with n when findings exist

SUPPRESSIONS

`cppcheck --suppress=<id> <path>`
Suppress an id everywhere

`cppcheck --suppress=<id>:<file> <path>`
Suppress an id in one file
*Wildcards *, ** and ? work in the file part.*

SUPPRESSIONS (cont.)

`cppcheck --suppress=<id>:<file>:<line> <path>`
Suppress at one line

`cppcheck --suppressions-list=<file> <path>`
Read suppressions from a text file
One per line, same format as --suppress. # and // start comments.

`cppcheck --inline-suppr <path>`
Honour cppcheck-suppress comments

`// cppcheck-suppress <id>`
Suppress on the next or same line

`// cppcheck-suppress [<id1>, <id2>]`
Suppress several ids at once

`// cppcheck-suppress-begin <id> ... -end <id>`
Suppress a block

`// cppcheck-suppress-file <id>`
Suppress for the whole file

ADDONS

`cppcheck --addon=<name>.py <path>`
Run a Python addon

`cppcheck --addon=<file>.json <path>`
Run an addon with settings
JSON keys: script, args, ctu.

`cppcheck --addon=misra.json --enable=style <file>`
Check MISRA C rules
Open-source coverage is partial. Rule texts come from MISRA.

RECIPES

CI gate on a CMake project

```
cmake -B build \
  -
DCMAKE_EXPORT_COMPILE_COMMANDS=
ON
cppcheck --project=build/
compile_commands.json \
  -ibuild --
enable=warning,performance \
  --error-exitcode=1 -j 4 --
inline-suppr
```

Fast incremental runs with a cache

```
mkdir -p .cppcheck-cache
cppcheck --cppcheck-build-
dir=.cppcheck-cache \
  --enable=style -j 4 -I include
src/
```