CMake Build system generator

CONCEPTS

CMake does not build anything itself: it generates files for a native tool (Make, Ninja, Visual Studio), which then compiles. Keep the build tree out of the source tree.

CMakeLists.txt project description
↓ `configure`
Build tree cache + generated files
↓ `--build`
Binaries via Make/Ninja
↓ `ctest / --install`
Tests, install results, prefix

Configure once, then build, test and install from the same tree

Cache variables (-D) persist in build/CMakeCache.txt, so later runs reuse them. Use --fresh or delete the build dir to reset.

CONFIGURE

`cmake -S <src> -B <build>`
Configure source into build dir

`cmake -B build -G "<generator>"`
Choose generator (Ninja, Unix Makefiles)

`cmake -B build -D<VAR>=<value>`
Set a cache entry

`cmake -B build -DCMAKE_BUILD_TYPE=<type>`
Debug, Release, RelWithDebInfo

`cmake -B build --fresh`
Drop the cache and reconfigure

`cmake -B build --toolchain <file>`
Use a toolchain file (cross-compile)

`-DCMAKE_EXPORT_COMPILE_COMMANDS=ON`
Write compile_commands.json

BUILD

`cmake --build <dir>`
Build everything

`cmake --build <dir> -j <N>`
Build with N parallel jobs

BUILD (cont.)

`cmake --build <dir> -t <target>`
Build one target

`cmake --build <dir> --clean-first`
Clean, then build

`cmake --build <dir> --config <cfg>`
Choose config (multi-config generators)

`cmake --build <dir> -v`
Show full compiler commands

INSTALL AND TEST

`cmake --install <dir>`
Install the project

`cmake --install <dir> --prefix <dir>`
Override the install prefix

`ctest --test-dir <dir>`
Run all tests

`ctest --test-dir <dir> --output-on-failure`
Show output of failing tests

`ctest -R <regex>`
Run tests whose name matches

`ctest -E <regex>`
Exclude matching tests

`ctest -j <N>`
Run tests in parallel

`ctest --rerun-failed`
Run only tests that failed last time

PRESETS

`cmake --list-presets`
List configure presets

`cmake --preset <name>`
Configure with a preset

`cmake --workflow --preset <name>`
Run configure, build, test in sequence

Presets live in CMakePresets.json (shared, committed) and CMakeUserPresets.json (personal, usually git-ignored).

INSPECT AND DEBUG

`cmake -L -N <dir>`
List cache variables, no reconfigure

`cmake --debug-find`
Show how find_package searches

CMAKELISTS.TXT

`cmake_minimum_required(VERSION <x.y>)`
Declare the minimum CMake version

`project(<name> VERSION <x.y> LANGUAGES CXX)`
Name the project

`add_executable(<tgt> <sources>)`
Define an executable target

`add_library(<tgt> STATIC|SHARED <sources>)`
Define a library target

`target_link_libraries(<tgt> PRIVATE <lib>)`
Link a dependency

`target_include_directories(<tgt> PUBLIC <dir>)`
Add include paths

`target_compile_features(<tgt> PUBLIC cxx_std_17)`
Require a language standard

`find_package(<Pkg> REQUIRED)`
Locate an installed package

`add_subdirectory(<dir>)`
Include a nested CMakeLists.txt

`enable_testing()`
Turn on ctest support

`add_test(NAME <t> COMMAND <exe>)`
Register a test

`install(TARGETS <tgt> DESTINATION bin)`
Install a target

Prefer target_* commands over global ones (include_directories, add_definitions). Use ${VAR} to read a variable and $<...> for generator expressions evaluated at build time.