

Building projects with CMake

Victor Eijkhout

Fall 2023



CMake is a portable build system that is becoming a *de facto* standard for C++ package management.
Also usable with C and Fortran.



1 Help! This software uses CMake!

- Using a cmake-based library

- Using CMake packages through pkgconfig

2 Help! I want to write CMake myself!

- Make your own CMake configuration

3 Help! I want people to use my CMake package!

- Making your package discoverable through pkgconfig

4 Example libraries

- Parallelism

5 TBB

- Data packages

- More libraries



Help! This software uses CMake!



Using a cmake-based library



- You have downloaded a library
- It contains a file `CMakeLists.txt`
- \Rightarrow you need to install it with CMake.
- ... and then figure out how to use it in your code.



- Use CMake for the configure stage, then make:

```
cmake -D CMAKE_INSTALL_PREFIX=/home/yourname/packages \  
      /home/your/software/package ## source location  
make  
make install
```

or

- do everything with CMake:

```
cmake ## arguments  
cmake --build ## stuff  
cmake --install ## stuff
```

We focus on the first option; the second one is portable to non-Unix environments.



1. The source directory is untouched
2. The build directory contains all temporaries
3. Your install directory (as specified to CMake) now contains executables, libraries, headers etc.

You can add these to `$PATH`, compiler options, `$LD_LIBRARY_PATH`.

But see later ...

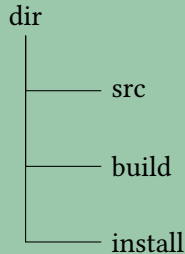
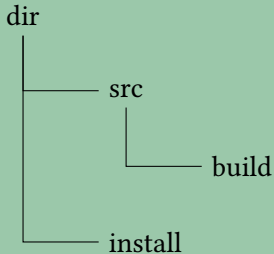


CMake creates makefiles;
makefiles ensure minimal required compilation

```
cmake          ## make the makefiles  
make           ## compile your project  
emacs onefile.c ## edit  
make          ## minimal recompile
```

Only if you add (include) files do you rerun CMake.





- In-source build: pretty common
- Out-of-source build: cleaner because never touches the source tree
- Some people skip the install step, and use everything from the build directory.



- Work from a build directory
- Specify prefix and location of CMakeLists.txt

```
1  ls some_package_1.0.0                # we are outside the source
2  ls some_package_1.0.0/CMakeLists.txt # source contains cmake file
3  mkdir builddir                        # location for temporaries
4  cd builddir                          # goto build location
5  cmake -D CMAKE_INSTALL_PREFIX=../installdir \
6      ../some_package_1.0.0            # cmake invocation
7  make                                 # make all tmp data in build loc
8  make install                         # move only final products
```



Download from

<https://eigen.tuxfamily.org/index.php>
and install.

What compiler is it finding? If you are at TACC, is it the module you have loaded?



Compiler settings:

```
cmake -D CMAKE_CXX_COMPILER=icpx
```

Alternatively:

```
export CXX=icpx  
cmake .....
```

Many settings can be done on the commandline:

```
-D BUILD_SHARED_LIBS=ON
```

Also check out the `ccmake` utility.



- CMake prints some sort of progress messages.
- To see commandlines:

```
cmake -D CMAKE_VERBOSE_MAKEFILE=ON ...  
make V=1
```

- CMake leaves behind a log and error file, but these are insufficient:
- ⇒ use the above verbose mode and capture all output.



Using CMake packages through pkgconfig



You have just installed a CMake-based library.
Now you need it in your own code, or in another library.
How easy can we make that?



You want to install an application/package
... which needs 2 or 3 other packages.

```
gcc -o myprogram myprogram.c \  
-I/users/my/package1/include \  
-L/users/my/package1/lib \  
-I/users/my/package2/include/package \  
-L/users/my/package2/lib64
```

or:

```
cmake \  
-D PACKAGE1_INC=/users/my/package1/include \  
-D PACKAGE1_LIB=/users/my/package1/lib \  
-D PACKAGE2_INC=/users/my/package2/include/package \  
-D PACKAGE2_LIB=/users/my/package2/lib64 \  
../newpackage
```

Can this be made simpler?



- Many packages come with a `package.pc` file
- Add that location to `PKG_CONFIG_PATH`
- The package can now be found by other CMake-based packages.



Let's say you've installed a library with CMake.
Somewhere in the installation is a `.pc` file:

```
find $TACC_SMTHNG_DIR -name \*.pc  
${TACC_SMTHNG_DIR}/share/pkgconfig/smithng3.pc
```

That location needs to be on the `PKG_CONFIG_PATH`:

```
export PKG_CONFIG_PATH=${TACC_SMTHNG_DIR}/share/pkgconfig:${  
    PKG_CONFIG_PATH}
```



Can you find the `.pc` file in the Eigen installation?



Packages with a `.pc` file can be found through the `pkg-config` command:

```
gcc -o myprogram myprogram.c \  
    $( pkg-config --cflags package1 ) \  
    $( pkg-config --libs package1 )
```

In a makefile:

```
CFLAGS = -g -O2 $$ ( pkg-config --cflags package1 )
```



```
#include "Eigen/Core"
int main(int argc, char **argv) {
    return 0;
}
```

Can you compile this on the commandline, using *pkg-config*? Small problem: 'eigen' wants to be called 'eigen3'.



You are installing a CMake-based library
and it needs Eigen, which is also CMake-based

1. you install Eigen with CMake, as above
2. you add the location of `eigen.pc` to `PKG_CONFIG_PATH`
3. you run the installation of the higher library:
this works because it can now find Eigen.



So how does a CMake install find libraries such as Eigen?

```
1  cmake_minimum_required( VERSION 3.12 )
2  project( eigentest )
3
4  find_package( PkgConfig REQUIRED )
5  pkg_check_modules( EIGEN REQUIRED eigen3 )
6
7  add_executable( eigentest eigentest.cxx )
8  target_include_directories(
9      eigentest PUBLIC
10     ${EIGEN_INCLUDE_DIRS})
```

Note 1: header-only so no library, otherwise `PACKAGE_LIBRARY_DIRS` and `PACKAGE_LIBRARIES` defined.

Note 2: you will learn how to write these configurations in the second part.



- You can use CMake to install libraries;
- You can use these libraries from commandline / makefile;
- You can let other CMake-based libraries find them.



Some packages come with `FindWhatever.cmake` or similar files.

Add package root to `CMAKE_MODULE_PATH`

Pity that there is not just one standard.

These define some macros, but you need to read the docs to see which.

Pity that there is not just one standard.

Some examples follow.



Help! I want to write CMake myself!



Make your own CMake configuration



You have a code that you want to distribute in source form for easy installation.

You decide to use CMake for portability.

You think that using CMake might make life easier.

⇒ To do: write the `CMakeLists.txt` file.



```
cmake_minimum_required( VERSION 3.12 )  
project( myproject VERSION 1.0 )
```

- Which cmake version is needed for this file?
(CMake has undergone quite some evolution!)
- Give a name to your project.
- Maybe pick a language.
C and C++ available by default, or:

```
enable_language(Fortran)
```

(list: *C, CXX, CSharp, CUDA, OBJC, OBJCXX, Fortran, HIP, ISPC, Swift*,
and a couple of variants of *ASM*)



- Declare a target: something that needs to be built, and specify what is needed for it

```
add_executable( myprogram )  
target_sources( myprogram PRIVATE program.cxx )
```

Use of macros:

```
add_executable( ${PROJECT_NAME} )
```

- Do things with the target, for instance state where it is to be installed:

```
install( TARGETS myprogram DESTINATION . )
```

relative to the prefix location.



Build an executable from a single source file:

```
cmake_minimum_required( VERSION 3.12 )  
project( singleprogram VERSION 1.0 )  
  
add_executable( program )  
target_sources( program PRIVATE program.cxx )  
install( TARGETS program DESTINATION . )
```




```
add_executable( myprogram myprogram.c myprogram.h )
```

Prefer 'target' design.



- Write a 'hello world' program;
- Make a CMake setup to compile and install it;
- Test it all.



This is a short program using Eigen:

```
#include <iostream>
#include <Eigen/Dense>

int main() {
    // Define a 3x3 matrix
    Eigen::Matrix3d matrix;
    matrix << 1, 2, 3,
              4, 5, 6,
              7, 8, 9;
    std::cout << "Original matrix:\n" << matrix << std::endl;
    return 0;
}
```

- Make a CMake setup to compile and install it;
- Test it.



First a library that goes into the executable:

```
add_library( auxlib )  
target_sources( auxlib PRIVATE aux.cxx aux.h )  
target_link_libraries( program PRIVATE auxlib )
```



Full configuration for an executable that uses a library:

```
1  cmake_minimum_required( VERSION 3.12 )
2  project( cmakeprogram VERSION 1.0 )
3
4  add_executable( program )
5  target_sources( program PRIVATE program.cxx )
6
7  add_library( auxlib )
8  target_sources( auxlib PRIVATE aux.cxx aux.h )
9
10 target_link_libraries( program PRIVATE auxlib )
11
12 install( TARGETS program DESTINATION . )
```

Library shared by default; see later.



In the configuration file:

```
add_library( auxlib STATIC )  
# or  
add_library( auxlib SHARED )
```

(default shared if left out), or by adding a runtime flag

```
cmake -D BUILD_SHARED_LIBS=TRUE
```

Build both by having two lines, one for shared, one for static.

Related: the `-fPIC` compile option is set by

`CMAKE_POSITION_INDEPENDENT_CODE`:

```
cmake -D CMAKE_POSITION_INDEPENDENT_CODE=ON
```



To have the library released too, use **PUBLIC**.
Add the library target to the **install** command.



```
1  cmake_minimum_required( VERSION 3.12 )
2  project( cmakeprogram VERSION 1.0 )
3
4  add_executable( program )
5  target_sources( program PRIVATE program.cxx )
6
7  add_library( auxlib STATIC )
8  target_sources( auxlib PRIVATE lib/aux.cxx lib/aux.h )
9
10 target_link_libraries( program PUBLIC auxlib )
11 target_include_directories( program PRIVATE lib )
12
13 install( TARGETS program DESTINATION bin )
14 install( TARGETS auxlib DESTINATION lib )
15 install( FILES lib/aux.h DESTINATION include )
```

Note the separate destination directories.



The previous setup was messy
Better handle the library through a recursive cmake
and make the usual `lib include bin` setup



Declare that there is a directory to do recursive make:

```
1  cmake_minimum_required( VERSION 3.13 )
2  # needs >3.12 to let the executable target find the .h file
3  project( cmakeprogram VERSION 1.0 )
4
5  add_executable( program )
6  target_sources( program PRIVATE program.cxx )
7  add_subdirectory( lib )
8  target_include_directories(
9      program PUBLIC lib )
10 target_link_libraries( program PUBLIC auxlib )
11 install( TARGETS program DESTINATION bin )
```

(Note that the name of the library comes from the subdirectory)



Installs into `lib` and `include`

```
1  cmake_minimum_required( VERSION 3.13 )
2  # needs >3.12 to let the executable target find the .h file
3
4  add_library( auxlib STATIC )
5  target_sources( auxlib
6      PRIVATE aux.cxx
7      PUBLIC aux.h )
8  install( TARGETS auxlib DESTINATION lib )
9  install( FILES aux.h DESTINATION include )
```



- Use `LD_LIBRARY_PATH`, or
- use `rpath`.

(Apple note: forced to use second option)

```
set_target_properties(  
    ${PROGRAM_NAME} PROPERTIES  
    BUILD_RPATH    "${CATCH2_LIBRARY_DIRS};${FMTLIB_LIBRARY_DIRS}"  
    INSTALL_RPATH  "${CATCH2_LIBRARY_DIRS};${FMTLIB_LIBRARY_DIRS}"  
)
```



```
include(ExternalProject)
ExternalProject_Add(googletest
  GIT_REPOSITORY    https://github.com/google/googletest.git
  GIT_TAG           master
  SOURCE_DIR        "${CMAKE_BINARY_DIR}/googletest-src"
  BINARY_DIR        "${CMAKE_BINARY_DIR}/googletest-build"
  CONFIGURE_COMMAND ""
  BUILD_COMMAND     ""
  INSTALL_COMMAND   ""
  TEST_COMMAND      ""
)
```



Help! I want people to use my CMake package!



Making your package discoverable through pkgconfig



Use the PKG_CONFIG_PATH variable:

```
$ module show cxxopts 2>&1 | grep -i pkg  
prepend_path("PKG_CONFIG_PATH", "/opt/cxxopts/intel23/lib64/pkgconfig")
```



configure_file line in CMakeLists.txt:

```
configure_file(  
    ${CMAKE_CURRENT_SOURCE_DIR}/${PROJECT_NAME}.pc.in  
    ${CMAKE_CURRENT_BINARY_DIR}/${PROJECT_NAME}.pc  
    @ONLY)
```



The .pc.in file:

```
prefix="@CMAKE_INSTALL_PREFIX@"  
exec_prefix="${prefix}"  
libdir="${prefix}/lib"  
includedir="${prefix}/include"  
  
Name: @PROJECT_NAME@  
Description: @CMAKE_PROJECT_DESCRIPTION@  
Version: @PROJECT_VERSION@  
Cflags: -I${includedir}  
Libs: -L${libdir} -l@libtarget@
```

Note the initial cap!

Combination of built-in variables and your own:

```
set( libtarget auxlib )
```



```
install(  
  FILES ${CMAKE_CURRENT_BINARY_DIR}/${PROJECT_NAME}.pc  
  DESTINATION share/pkgconfig  
)
```



Example libraries



Parallelism



MPI has a module:

```
find_package( MPI )  
target_include_directories(  
    ${PROJECT_NAME} PUBLIC  
    ${MPI_C_INCLUDE_DIRS} )  
target_link_libraries(  
    ${PROJECT_NAME} PUBLIC  
    ${MPI_C_LIBRARIES} )
```



```
find_package( MPI )
target_include_directories(
    ${PROJECT_NAME} PUBLIC
    ${MPI_CXX_INCLUDE_DIRS} )
target_link_libraries(
    ${PROJECT_NAME} PUBLIC
    ${MPI_CXX_LIBRARIES} )
```



```
find_package( MPI )
target_include_directories(
    ${PROJECT_NAME} PUBLIC
    ${MPI_INCLUDE_DIRS} )
target_link_directories(
    ${PROJECT_NAME} PUBLIC
    ${MPI_LIBRARY_DIRS} )
target_link_libraries(
    ${PROJECT_NAME} PUBLIC
    ${MPI_Fortran_LIBRARIES} )
```




```
if( MPI_Fortran_HAVE_F08_MODULE )
else()
    message( FATAL_ERROR "No f08 module for this MPI" )
endif()
```



```
find_package( mpl REQUIRED )
target_include_directories(
    ${PROJECT_NAME} PUBLIC
    ${CMAKE_CURRENT_SOURCE_DIR}
    mpl::mpl )
target_link_libraries(
    ${PROJECT_NAME} PUBLIC
    mpl::mpl )
```



```
find_package(OpenMP)  
target_link_libraries(  
    ${PROJECT_NAME}  
    PUBLIC OpenMP::OpenMP_C )
```



```
find_package(OpenMP)
if(OpenMP_CXX_FOUND)
else()
    message( FATAL_ERROR "Could not find OpenMP" )
endif()
target_link_libraries(
    ${PROJECT_NAME}
    PUBLIC OpenMP::OpenMP_CXX )
```



```
enable_language(Fortran)  
find_package(OpenMP)  
target_link_libraries(  
    ${PROJECT_NAME}  
    PUBLIC OpenMP::OpenMP_Fortran )
```



TBB



```
find_package(TBB REQUIRED)  
target_link_libraries( ${PROJECT_NAME} PUBLIC TBB::tbb)
```



```
find_package(Kokkos REQUIRED)  
target_link_libraries(myTarget Kokkos::kokkos)
```

Either set `CMAKE_PREFIX_PATH` or add

```
-DKokkos_ROOT=<Kokkos Install Directory>/lib64/cmake/Kokkos
```

Maybe:

```
-DCMAKE_CXX_COMPILER=<Kokkos Install Directory>/bin/  
  nvcc_wrapper
```

See <https://kokkos.org/kokkos-core-wiki/ProgrammingGuide/Compiling.html>



Data packages



C:

```
find_package( PkgConfig REQUIRED )  
pkg_check_modules( HDF REQUIRED hdf5 )  
message( STATUS "Hdf5 includes: ${HDF_INCLUDE_DIRS}" )  
  
target_include_directories(  
    ${PROJECTNAME} PUBLIC  
    ${HDF_INCLUDE_DIRS}  
)
```



C:

```
find_package( PkgConfig REQUIRED )
pkg_check_modules( NETCDF REQUIRED netcdf )

target_include_directories(
    ${PROJECTNAME} PUBLIC
    ${NETCDF_INCLUDE_DIRS} )
target_link_libraries(
    ${PROJECTNAME} PUBLIC
    ${NETCDF_LIBRARIES} )
target_link_directories(
    ${PROJECTNAME} PUBLIC
    ${NETCDF_LIBRARY_DIRS} )
target_link_libraries(
    ${PROJECTNAME} PUBLIC netcdf )
```



```
find_package( PkgConfig REQUIRED )
pkg_check_modules( NETCDFFF REQUIRED netcdf-fortran )
pkg_check_modules( NETCDF REQUIRED netcdf )

target_include_directories(
    ${PROJECTNAME} PUBLIC
    ${NETCDFFF_INCLUDE_DIRS}
)

target_link_libraries(
    ${PROJECTNAME} PUBLIC
    ${NETCDFFF_LIBRARIES} ${NETCDF_LIBRARIES}
)

target_link_directories(
    ${PROJECTNAME} PUBLIC
    ${NETCDFFF_LIBRARY_DIRS} ${NETCDF_LIBRARY_DIRS}
)

target_link_libraries(
    ${PROJECTNAME} PUBLIC netcdf )
```



Third party C++ interface to hdf5

```
find_package( HighFive REQUIRED )  
target_link_libraries( ${PROJECTNAME} HighFive)
```



More libraries



Package dependent:

- Sometimes through `pkg-config`:
find the `.pc` file
- Sometimes through a `Find...` module
see CMake documentation



```
find_package( PkgConfig REQUIRED )
pkg_check_modules( CATCH2 REQUIRED catch2-with-main )
target_include_directories(
    ${PROGRAM_NAME} PUBLIC
    ${CATCH2_INCLUDE_DIRS}
)
target_link_directories(
    ${PROGRAM_NAME} PUBLIC
    ${CATCH2_LIBRARY_DIRS}
)
target_link_libraries(
    ${PROGRAM_NAME} PUBLIC
    ${CATCH2_LIBRARIES}
)
```



Header-only:

```
find_package( PkgConfig REQUIRED )  
pkg_check_modules( OPTS REQUIRED cxxopts )  
target_include_directories(  
    ${PROGRAM_NAME} PUBLIC  
    ${OPTS_INCLUDE_DIRS}  
)
```



Header-only:

```
cmake_minimum_required( VERSION 3.12 )  
project( eigentest )  
  
find_package( PkgConfig REQUIRED )  
pkg_check_modules( EIGEN REQUIRED eigen3 )  
  
add_executable( eigentest )  
target_sources( eigentest PRIVATE eigentest.cxx )  
target_include_directories(  
    myprogram PUBLIC  
    ${EIGEN_INCLUDE_DIRS})
```



```
find_package( PkgConfig REQUIRED )
pkg_check_modules( FMTLIB REQUIRED fmt )
target_include_directories(
    ${PROGRAM_NAME} PUBLIC ${FMTLIB_INCLUDE_DIRS}
)
target_link_directories(
    ${PROGRAM_NAME} PUBLIC ${FMTLIB_LIBRARY_DIRS}
)
target_link_libraries(
    ${PROGRAM_NAME} PUBLIC ${FMTLIB_LIBRARIES}
)
```



Has its own module:

```
find_package( range-v3 REQUIRED )  
target_link_libraries(  
    ${PROGRAM_NAME} PUBLIC range-v3::range-v3 )
```

