

# Help! This software uses 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.)

Many libraries can be installed with CMake.



1 Using a cmake-based library

2 Using CMake packages through pkgconfig



# 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 ...



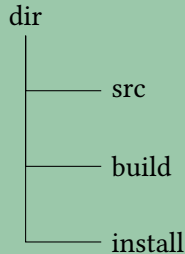
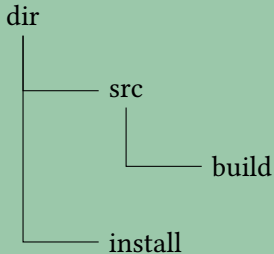
CMake creates makefiles;  
makefiles ensure minimal required compilation

```
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
```



Compiler settings:

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

Alternatively:

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

Many other settings done in the *CMakeLists.txt*

```
-D BUILD_SHARED_LIBS=ON
```



- 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:
- $\Rightarrow$  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/packaage \  
-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/packaage \  
-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}
```



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 )
```



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.

