Help! This software uses CMake!

Victor Eijkhout

Fall 2023

Justification

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.

Table of contents

Using a cmake-based library

What are we talking here?

- 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.

Building with CMake

- Use CMake for the 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:

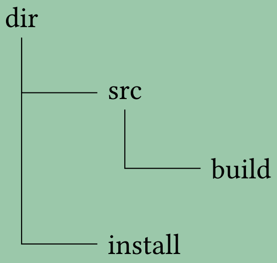
```
cmake ## arguments
cmake --build ## stuff
cmake --install ## stuff
```

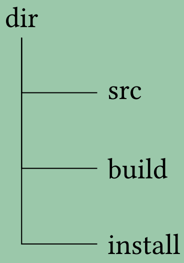
We focus on the first option; the second one is portable to non-Unix environments.

What does this buy you?

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 . . .

Directory structure: two options





- 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.

Out-of-source build: preferred

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

```
ls some_package_1.0.0 # we are outside the source
ls some_package_1.0.0/CMakeLists.txt # source contains cmake file
mkdir builddir && cd builddir # goto build location
cmake -D CMAKE_INSTALL_PREFIX=../installdir \
      ../some_package_1.0.0
make # make all tmp data in build loc
make install # move final products to install loc
```

Using packages through pkgconfig

What are we talking here?

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?

Problem

You want to install a 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?

Finding packages with 'pkg config'

- 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.

Somewhere in the installation is a `.pc` file:

```
find $TACC_EIGEN_DIR -name \*.pc
${TACC_EIGEN_DIR}/share/pkgconfig/eigen3.pc
```

That location needs to be on the `PKG_CONFIG_PATH`:

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

Scenario 1: finding without cmake

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 $( shell pkg-config --cflags package1 )
```

Scenario 2: finding from CMake

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.

Lifting the veil

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

```
cmake_minimum_required( VERSION 3.12 )
project( eigentest )

find_package( PkgConfig REQUIRED )
pkg_check_modules( EIGEN REQUIRED eigen3 )

add_executable( eigentest eigentest.cxx )
target_include_directories(
        eigentest PUBLIC
        ${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 configuration in the second part.