Building projects with CMake

Victor Eijkhout

Fall 2022

Justification

CMake is a portable build system that is becoming a *de facto* standard for C++ package management.

Also usable with C and Fortran.

1 Building software the old way

Using ‘GNU Autotools’:

```
./configure
make
make install
```

2 User vs system packages

The `make install` often tries to copy to a system directory. If you're not the admin, do:

```
./configure --prefix=/home/yourname/mypackages
```

with a location of your choice.

3 Building with CMake

- Either replace only the configure stage

```
cmake ## argments
make
make install
```

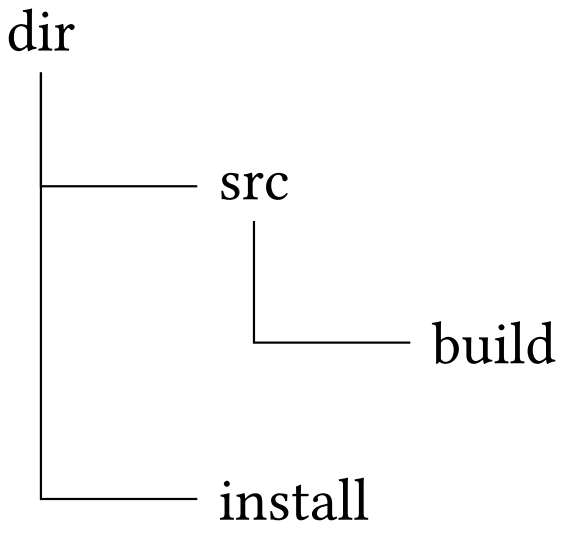
or

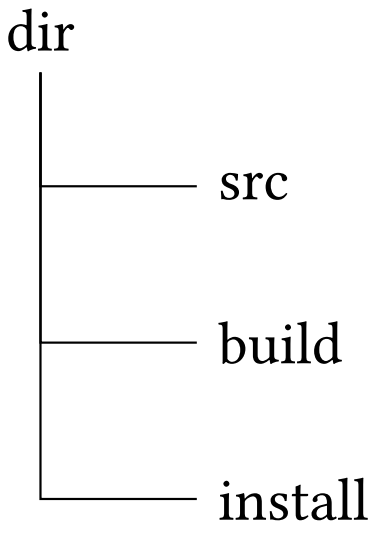
- do everything with CMake:

```
cmake ## argments
cmake --build ## stuff
cmake --install ## stuff
```

(The second one is portable to non-Unix environments.)

4 Directory structure





- In-source build: pretty common
- Out-of-source build: cleaner because never touches the source tree

5 Out-of-source build

- 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 install
```

6 The CMakeLists file

- Which cmake version is needed for this file?
  (CMake has undergone quite some evolution!)
- Give a name to your project.

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

7 Target philosophy

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

```
add_executable( myprogram program.cxx )
install( TARGETS myprogram DESTINATION . )
```

Use of macros:

```
add_executable( ${PROJECT_NAME} program.cxx )
```

8 Example: single source

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

add_executable( program program.cxx )
install( TARGETS program DESTINATION . )
```

9 Use of a library

First a library that goes into the executable:

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

10 Example: library during build

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

add_executable( program program.cxx )
add_library( auxlib
             aux.cxx aux.h )
target_link_libraries( program PRIVATE auxlib )
install( TARGETS program DESTINATION . )
```

11 Release a library

To have the library released too use **`PUBLIC`**.

Add the library target to the **`install`** command.

12 Example: released library

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

add_executable( program program.cxx )
add_library( auxlib
             aux.cxx aux.h )
target_link_libraries( program PUBLIC auxlib )
install( TARGETS program auxlib DESTINATION . )
```

13 Finding packages with ‘pkg config’

- Many packages come with a `package.pc` file
- Add that location to `PKG_CONFIG_PATH`
- That defines variables in your own cmake file

Example: PETSc

add `$PETSC_DIR/$PETSC_ARCH/lib/pkgconfig` to config path, then

```
find_package( PkgConfig REQUIRED )
pkg_check_modules( PETSC REQUIRED petsc )
target_include_directories(
        program PUBLIC
        ${PETSC_INCLUDE_DIRS} )
```