make, schmake: CMake

Stewart A. Levin and Robert G. Clapp

ABSTRACT

This year SEP has switched from GNU *automake/autoconf/libtool* builds of our public source tree to *CMake*. Here we highlights the advantages, and annoyances, of using CMake and an assessment of whether it brings significant advantages to our current makefile-based seismic processing and inversion environment.

INTRODUCTION

Since as least as far back as 2010, our public SEPlib source distribution has been configured and built with the sophisticated GNU portability tools *automake*, *autoconf* and, more recently, *libtool*. These let the end user build and install our distributions by running `./configure` in the top level of the source tree followed by `make && make install`. Those tools, however, are unwieldy to configure and require fairly intimate knowledge of close to a thousand pages of documentation at `gnu.org/software` to deploy consistently and correctly. They also have a lot of overhead—it takes over an hour to build SEPlib from scratch.

About five years ago, co-author Clapp, began using *CMake* to build his own software, most notably his QuickTime-based 3D seismic viewer. *CMake* is designed around modern programming paradigms such as plugins (modules) and object-oriented (C++) features. Over that period he ported the SEPlib source build system to *CMake* as well. This proved remarkably easy and co-author Levin was able to make additions, deletions, and edits, despite substandard documentation, much more easily than with the GNU-based system. Additionally, a bottom up SEPlib build takes only ten minutes.

At this juncture, CMake-based SEPlib distribution is basically ready for prime time and will be the default build system for the forseeable future. This raises some interesting issues and questions.

ISSUES AND QUESTIONS

Warts *aka* novice usage

CMake is the joint development of a large community of developers, most volunteer, and, as such, contains a vast number of features that the ordinary user, and

even more so the novice, would not typically be using. This panoply can be quite daunting and is only partially alleviated by the basic seven step tutorial provided at `http://cmake.org`. Such issues include:

- How to specify output and installation directories.
- How to specify compilers and linkers and their options.
- How to define and use CMake variables.
- Where to find such information about CMake.

For the more sophisticated, the book **Mastering CMake** (Martin and Hoffman, 2015) is intended to provide authoritative information on *CMake*. While it does so to a great extent, more than half of it consists of reprints of a tutorial and manual pages from `cmake.org` and its index is almost useless for searching. Frankly not worth the 50 buck price. We have mostly resorted to the alternative of finding examples elsewhere that are similar to what we want to accomplish and tailoring them to the task at hand.

Software distribution

One of the requirements for using CMake in our software distribution is the need to have the `cmake` utility on the receiving end. The GNU setup only required `/bin/sh` or an equivalent such as `bash` in order to run the `configure` script plus a reasonably modern version of `make` such as `gmake` to build and install the libraries and executables. In the near future we will use the *CPack* utility to build one or more binary installers for SEPlib. While this will not obviate the need for *CMake* for system development, it will allow precompiled executables to be conveniently distributed.

Fortunately, SEP research will soon be distributed via `docker` (Clapp, 2016) files which will contain `cmake` and can be run safely and securely on most any Intel-compatible computer architecture.

SEP data processing

SEP goes to great lengths to meet the gold standard of research reproducibility: "Burn, Build, View." That is, whenever possible, all results and intermediate files can be deleted, a processing sequence rerun automatically, and the resulting output identical to the results in the research report. This is done at the Stanford Exploration Project by using makefiles to specify inputs, dependencies, outputs, and command lines that transform inputs into outputs. For example, the makefile excerpt

```
output.H : input.H process.par
           $(SEP)/bin/Filter < input.H > output.H par=process.par
```

generates `output.H` from `input.H` using the SEP `Filter` program with keyword parameters given in the `process.par` parameter file. An equivalent in cmake syntax is

```
add_custom_command(
   OUTPUT output.H
   DEPENDS input.H process.par
   COMMAND ${SEP}/bin/Filter < input.H par=process.par > output.H
   )
```

which is somewhat more verbose. It is also subtly different. First, this is not executed by `cmake`. `cmake` simply converts it into a makefile and ancillary files. The user still has to type `make output.H` in order to run the commands. Second, the `output.H` file is marked as GENERATED which means that a subsequent `make clean` will remove it. This relieves the user from explicitly listing intermediate and final outputs individually in a makefile "burn" rule. Third, there is an automatic dependency of `output.H` on the cmake input (`CMakeLists.txt`) so that if the command or various inputs are modified, `cmake` is rerun automatically to create a new makefile.

DISCUSSION

Both theoretically and practically, albeit with rather inadequate reference documentation, *CMake* offers significant advantages in ease of use and efficiency in source code builds and installs. For reproducible research, there are advantages and disadvantages. On the plus side, cmake syntax is more verbose, hence more readable, than makefile syntax. On the minus side, the default GENERATED property[1] presents the hazard of deleting intermediate or final outputs that are only conditionally reproducible whether for lack of access to its inputs or the cost of recomputing key results.

REFERENCES

Clapp, R. G., 2016, Reproducibility through containers : SEP-Report, **165**, 193–198.
Martin, K. and B. Hoffman, 2015, Mastering CMake Version 3.1: Kitware Inc.

[1]Yes it can be unset, but the documentation is silent on this.