LA-UR-16-24845
Approved for public release; distribution is unlimited.

Title: Software Build and Delivery Systems

Author(s): Robey, Robert W.

Intended for: Report

Issued: 2016-07-12

Software Build and Delivery Systems

Bob Robey

July 10th, 2016

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

Hierarchy of Software Build and Delivery Systems

- **Make – perform a sequence of operations to build a product**
- **Configure – provides system independence**
  - autoconf, cmake
- **Package manager – automatic installation of software**
- **Software Containers – Docker and many others**

These have been heavily developed in the open source community and should be leveraged – don't reinvent!

One of the goals is to maximize the success rate of new users and developers when first trying your software.

- **First impressions are important.**
- **Early successes are important.**
- **This also reduces critical documentation costs.**

Make[1]

- **Make is a very unique "programming language"**
  - Builds a dependency graph (DAG[1]) of commands informed by time stamps
  - Compares targets (products) and sources (dependencies) timestamps; command is executed if target is older than sources
  - Skips unnecessary work – can greatly reduce build times
- **Too much trouble to use a makefile?**
  - The effort will payback in very short order, especially as program size grows.

[1]DAG – directed acyclic graph, fancy term to say there is always a proper order that can be found to execute commands without circular paths

[1]Stuart Feldman, Bell Labs, April 1976

Make

- **Convention is to put commands in a file called “Makefile”**
  - This is so that it shows up first in listings. (MAKEFILE and makefile are also default names; -f <file> also allows any name
- **Make can be used for documentation and many other code development tasks in addition to compiling.**

Make – Basic structure

- **Basic structure**

  target: dependency list

  <tab> command

Example:

```
myprog:  main.o sub.o
    $(CC) $(LDFLAGS) –o $@ $^ $(LIBS)
sub.o: sub.c
    $(CC) $(CFLAGS) –c sub.c
clean:
    rm -f *.o; rm –f myprog
%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@
```

$^ -- dependencies, $@ -- target, $< first dependency

Make (continued)

A lot can be done just by adding to the standard macros

- CFLAGS = $(CFLAGS) –O3

Comments start with # character

Include files with

- include make.in

\ -- Line continuation (no space allowed after \)

Game of Life Example

Convert the following script into a makefile. Include a build, run and clean command

```
#!/bin/csh
#module load openmpi
#module load gcc
echo compiling game of life
rm iter*
mpicc -o gameOfLife gameOfLife.c
mpirun -np 4 gameOfLife
echo done
```

Makefile example – see Makefile.example in gol_MPI directory

```
CFLAGS = -I$(MPI_INCLUDE)
LIBS = -L$(MPI_LIB) –lmpi

gameOfLife: gameOfLife_mpi.o
    $(CC) $(LDFLAGS) –o $@ $^ $(LIBS)

run: gameOfLife
    mpirun –np 4 gameOfLife
    echo done

clean:
    rm –f iter* gameOfLife *.o
```

System Portability

- **The standard practice up until the early 1990s was to have a bunch of different make<system>.inc files for all possible systems. This became a maintenance nightmare and so –**
- **Several different improvements emerged**
  - metaconfig – Larry Wall and Perl. Autodetected some configure parameters and prompted for others. This had the problem that most users had no idea what to put in for the prompts
  - configure – part of gcc by Richard Stallman, and Cygnus
  - imake – part of X11. It was supposed to depend of parameters set in system level files, but these parameters specifying locations of things were seldom set.
  - Autoconf – by David McKensie, 1991, tested for features in real-time, thus having a chance of working on an unknown system and avoiding the need for a database of system information

Autoconf, Automake and libtools

- **Autoconf proved to be superior and is now used in nearly all open source packages in popular distributions**
- **configure.ac, Makefile.am, and config.h.in are used to create a Makefile, configure 'script', and a config.h file.**
- **The usual process is to:**

  ./configure <options>

  make

Common options are

- **--enable-debug**
- **--prefix=<install location>**

Makefile.am

```
all: gameOfLife
default: gameOfLife

bin_PROGRAMS = gameOfLife

gameOfLife_C_SRCS = gameOfLife_mpi.c

gameOfLife_SOURCES = ${gameOfLife_C_SRCS}

run: gameOfLife
	mpirun -np 4 gameOfLife
	echo done

distclean:
	rm -rf iter*
```

configure.ac – part 1 of 3

```
AC_INIT(gameOfLife, v1.0)
AC_CONFIG_AUX_DIR(./config)
AM_INIT_AUTOMAKE([foreign])
define([AC_CACHE_LOAD], )dnl
define([AC_CACHE_SAVE], )dnl

AC_ARG_ENABLE( debug,
 [--enable-debug                  - Compiles code with debug flags ],
 )

AC_ARG_ENABLE( mpi,
 [--enable-mpi                    - Compiles code with mpi library ],
 enable-mpi=yes)

# Checks for programs
AC_PROG_CC
AM_PROG_CC_C_O
AC_PROG_INSTALL
AC_PROG_MAKE_SET
AC_PROG_RANLIB
```

configure.ac – part 2 of 3

```
AC_CANONICAL_HOST
case "$host" in
   *linux*)
      host=linux
      CFLAGS="-std=gnu99"
      if test "${enable_debug}" = "yes"; then
        CFLAGS="-g -DDEBUG=1 ${CFLAGS}"
      else
        CFLAGS="-g -O3 ${CFLAGS}"
      fi
      RPATH_PREFIX="-Wl,“
      ;;
```

```
   *darwin*)
      host=darwin
      CFLAGS="-std=gnu99"
      if test "${enable_debug}" = "yes"; then
        CFLAGS="-g -DDEBUG=1 ${CFLAGS}"
      else
        CFLAGS="-g -O3 ${CFLAGS}"
      fi
      RPATH_PREFIX="-Wl,“
      ;;
    *)
      echo "Unknown machine"
      exit 1
      ;;
esac
```

configure.ac – part 3 of 3

```
if test "${enable_mpi}" != "no"; then
  AC_PATH_PROG(MPI_BIN, mpicc)
  FOUND_MPI="no"
  if test ! -z "${MPIHOME}"; then
    AC_CHECK_FILE(${MPIHOME}/include/mpi.h,
                  AM_CPPFLAGS="${AM_CPPFLAGS} -I${MPIHOME}/include" && FOUND_MPI="yes")
    LIBS="-L${MPIHOME}/lib -lmpi"
    if test "${RPATH_LIBS}x" = "x"; then
      RPATH_LIBS="${MPIHOME}/lib"
    else
      RPATH_LIBS="${RPATH_LIBS}:${MPIHOME}/lib"
    fi

    if test "${FOUND_MPI}" = "yes"; then
      AC_DEFINE([USE_MPI],"yes","Use MPI for parallel code")
    fi
  fi
fi
```

```
if test ! -z "${RPATH_LIBS}"; then
   LIBS="${LIBS} ${RPATH_PREFIX}-rpath ${RPATH_PREFIX}${RPATH_LIBS}"
fi

AC_SUBST(CC)
AC_SUBST(CFLAGS)
AC_SUBST(AM_CPPFLAGS)
AC_SUBST(LIBS)

AC_CONFIG_FILES([Makefile])

AC_OUTPUT
```

rpath – run-time search path hard-coded into executable

- Note the use of rpath in the link line for the build
- Syntax is –Wl,-rpath (-Wl means to pass to the linker)
- The rpath compile option should always be used in a “module” environment where users may have many different versions of a compiler or library.
- The rpath also provides additional security by avoiding loading of trojan libraries in a user’s LD_LIBRARY_PATH.

Autoconf example

- **Look at configure.ac and Makefile.am in gol_MPI directory**
- **Setup autoconf to run**
  - aclocal
  - mkdir config
  - automake --add-missing
  - autoconf
- **Run configure**
  - ./configure
  - Make
- **Try changing compiler and mpi versions and see if it works**

cmake

- Cmake is a newer alternative to autoconf that is easier to setup for most projects. It has standard “find” rules for most common packages.
- Originated around 2000 by Bill Hoffman, Kitware and is integral to LANL’s Paraview project
- It is not as easy to add more rules to cmake – at least for those of us used to “make” syntax

Simple Cmake project -- CMakeLists.txt

```
cmake_minimum_required (VERSION 2.8.11)
project (gameOfLife)
find_package(MPI)
if (MPI_FOUND)
   include_directories(${MPI_INCLUDE_PATH})
endif (MPI_FOUND)
add_executable (gameOfLife gameOfLife_mpi.c)
set_target_properties(gameOfLife PROPERTIES COMPILE_DEFINITIONS HAVE_MPI)
target_link_libraries(gameOfLife ${MPI_LIBRARIES})
```

Additional CMakeLists.txt commands (before add_executable) – optional versions, debug, and opt

```
# The version number.
set (gameOfLife_VERSION_MAJOR 2)
set (gameOfLife_VERSION_MINOR 0.7)
set (gameOfLife_VERSION "${gameOfLife_VERSION_MAJOR}.${gameOfLife_VERSION_MINOR}")
if (NOT CMAKE_BUILD_TYPE)
  set(CMAKE_BUILD_TYPE RelWithDebInfo)
endif(NOT CMAKE_BUILD_TYPE)
if ("${CMAKE_C_COMPILER_ID}" STREQUAL "Clang")
  # using Clang
  set(CMAKE_C_FLAGS_DEBUG "${CMAKE_C_FLAGS_DEBUG} -Wall -g -O0 -Wextra")
  set(CMAKE_C_FLAGS_RELEASE "${CMAKE_C_FLAGS_RELEASE} -g -O3")
elseif ("${CMAKE_C_COMPILER_ID}" STREQUAL "GNU")
  # using GCC
  set(CMAKE_C_FLAGS_DEBUG "${CMAKE_C_FLAGS_DEBUG} -Wall -g -O0 -Wextra")
  set(CMAKE_C_FLAGS_RELEASE "${CMAKE_C_FLAGS_RELEASE} -g -O3")
elseif ("${CMAKE_C_COMPILER_ID}" STREQUAL "Intel")
  # using Intel C++
  set(CMAKE_C_FLAGS_DEBUG "${CMAKE_C_FLAGS_DEBUG} -Wall -g -O0 -Wextra")
  set(CMAKE_C_FLAGS_RELEASE "${CMAKE_C_FLAGS_RELEASE} -g -O3")
endif()
```

Using CMakeLists.txt

cmake .

make

To clean up from build

rm -rf CMakeCache.txt CMakeFiles cmake_install.cmake Makefile

See cmake/Modules for sample custom cmake commands to do a distclean and to find OpenCL library and include files

Project documentation

- **Sphinx – builds pdf, html, and epub versions**
  - Uses markdown language, ReSTructured text
  - Can handle latex in documentation source
  - Interfaces with Doxygen to extract reference guide material from source via the *breathe* package

- **Pandoc – alternative package to generate multiple document formats**

Sphinx

- Load python-epd on moonlight or wolf
- make docs subdirectory in your code and then run
- sphinx-quickstart and answer questions (generally with defaults)
- Run “make latexpdf”, “make html” or “make singlehtml”’
- View pdf in _build/latex/<name>.pdf
- To integrate with cmake, in the directory above docs, add the cmake commands on the next slide

- See http://www.sphinx-doc.org/en/stable/tutorial.html for more on how to write documentation

Adding documentation – sphinx build targets in cmake

```
set(DOC_SRCS mocs/index.rst docs/About.rst docs/GettingStarted.rst
        docs/DevelopersGuide.rst docs/UsersGuide.rst)
set(PDFDOC_SRCS docs/indexpdf.rst docs/About.rst docs/GettingStarted.rst
        docs/DevelopersGuide.rst docs/UsersGuide.rst)

if (${CMAKE_PROJECT_NAME} MATCHES ${PROJECT_NAME})
  set(doc_prefix "")
else (${CMAKE_PROJECT_NAME} MATCHES ${PROJECT_NAME})
  set(doc_prefix gameOfLife_)
endif (${CMAKE_PROJECT_NAME} MATCHES ${PROJECT_NAME})

add_custom_target(${doc_prefix}doc
              COMMAND make ${doc_prefix}pdfdoc
              COMMAND make ${doc_prefix}htmldoc
              COMMAND make ${doc_prefix}singlehtmldoc)
set_target_properties(${doc_prefix}doc PROPERTIES
EXCLUDE_FROM_ALL TRUE)

add_custom_command(OUTPUT _build/latex/gameOfLife.pdf
               WORKING_DIRECTORY $
{CMAKE_CURRENT_BINARY_DIR}/docs
               COMMAND make latexpdf >& pdfdoc.out
               DEPENDS ${PDFDOC_SRCS})
add_custom_target(${doc_prefix}pdfdoc DEPENDS _build/latex/
gameOfLife.pdf)
set_target_properties(${doc_prefix}singlehtmldoc PROPERTIES
EXCLUDE_FROM_ALL TRUE)
```

```
add_custom_command(OUTPUT _build/html/UsersGuide.html
               WORKING_DIRECTORY $
{CMAKE_CURRENT_BINARY_DIR}/docs
               COMMAND make html >& htmldoc.out
               DEPENDS ${DOC_SRCS})
add_custom_target(${doc_prefix}htmldoc DEPENDS _build/html/
UsersGuide.html)
set_target_properties(${doc_prefix}htmldoc PROPERTIES
EXCLUDE_FROM_ALL TRUE)

add_custom_command(OUTPUT _build/singlehtml/index.html
               WORKING_DIRECTORY $
{CMAKE_CURRENT_BINARY_DIR}/docs
               COMMAND make singlehtml >& singlehtmldoc.out
               DEPENDS ${DOC_SRCS})
add_custom_target(${doc_prefix}singlehtmldoc DEPENDS _build/
singlehtml/index.html)
set_target_properties(${doc_prefix}singlehtmldoc PROPERTIES
EXCLUDE_FROM_ALL TRUE)
```

Package Managers and Spack

Ian Murdock has commented that package management is "the single biggest advancement Linux has brought to the industry", that it blurs the boundaries between operating system and applications, and that it makes it "easier to push new innovations [...] into the marketplace and [...] evolve the OS". (from Wikipedia on package managers)

Many package managers have been developed since Redhat's package manager (rpm) and later when Debian perfected it. Spack is targeted at the HPC community.

To install a package, "*spack install paraview*" will build paraview, first building and installing all the dependencies

Package Manager – Spack example

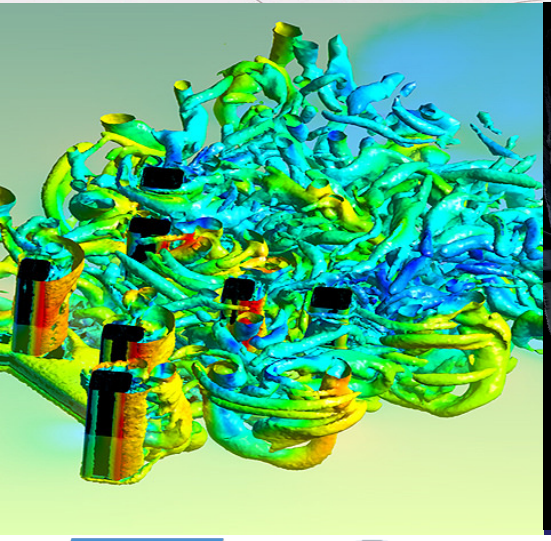
- **Each package has a set of instructions to build and install. For zlib:**

```
from spack import *

class Zlib(Package):
    """zlib is designed to be a free, general-purpose, legally unencumbered --
       that is, not covered by any patents -- lossless data-compression library for
       use on virtually any computer hardware and operating system.
    """

    homepage = "http://zlib.net"
    url      = "http://zlib.net/zlib-1.2.8.tar.gz"

    version('1.2.8', '44d667c142d7cda120332623eab69f40')

    def install(self, spec, prefix):
        configure("--prefix=%s" % prefix)

        make()
        make("install")
```

Docker and Software Containers

- Docker can package an application and its dependencies in a virtual container that can be deployed to a cloud server.
- Released in March, 2013, Docker popularized the software container concept that had been around since at least 2000 in forms such as BSD jails.

- The CLAMR proxy app was recently deployed and tested in a software container.



Los Alamos
NATIONAL LABORATORY

EST.1943

Delivering science and technology
to protect our nation
and promote world stability

Parallel Computing SRI

ISTI
INFORMATION SCIENCE & TECHNOLOGY INSTITUTE

NSEC

NNSA
National Nuclear Security Administration

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA