Building projects with make and cmake

CS-210: Introduction to Unix

Robert Bruce

Makefiles: What's the motivation?

- Open-source projects are often available with source code only. You must build the application for your specific machine architecture.
- Open-source projects frequently contain many files which must be individually compiled before being linked together into an application. The files that are linked together are often dependency files or open-source libraries that the project requires in order to run.
- Makefiles are the instructions for how to effectively build a massive project into an application.
- Building complex applications in the tech industry – especially in teams – is frequently done with Makefiles and variants such as Cmake.
- Knowing how to create and build with Makefiles is an essential skill Computer Science majors must acquire.

Dissecting a Makefile

```
CC = gcc
# define any compile-time flags
CFLAGS =

# define any directories containing header files other than /usr/include
INCLUDES =

# define library paths in addition to /usr/lib
#   if I wanted to include libraries not in /usr/lib I'd specify their path using -Lpath
LFLAGS =

# define any libraries to link into executable:
LIBS =

# define the C source files
SRCS =

# define the C object files
#
# This uses Suffix Replacement within a macro:
#   $(name:string1=string2)
#         For each word in 'name' replace 'string1' with 'string2'
# Below we are replacing the suffix .c of all words in the macro SRCS
# with the .o suffix
#
OBJS = $(SRCS:.c=.o)

# define the executable file
MAIN =
```

Dissecting a Makefile: SRCS specifies the source file or files to compile

```
CC = gcc
# define any compile-time flags
CFLAGS =

# define any directories containing header files other than /usr/include
INCLUDES =

# define library paths in addition to /usr/lib
#   if I wanted to include libraries not in /usr/lib I'd specify their path using -Lpath
LFLAGS =

# define any libraries to link into executable:
LIBS =

# define the C source files
SRCS =

# define the C object files
#
# This uses Suffix Replacement within a macro:
#   $(name:string1=string2)
#         For each word in 'name' replace 'string1' with 'string2'
# Below we are replacing the suffix .c of all words in the macro SRCS
# with the .o suffix
#
OBJS = $(SRCS:.c=.o)

# define the executable file
MAIN =
```

Dissecting a Makefile: SRCS specifies the source file or files to compile

```
CC = gcc
# define any compile-time flags
CFLAGS =

# define any directories containing header files other than /usr/include
INCLUDES =

# define library paths in addition to /usr/lib
#   if I wanted to include libraries not in /usr/lib I'd specify their path using -Lpath
LFLAGS =

# define any libraries to link into executable:
LIBS =

# define the C source files
SRCS =  project_file1.c project_file2.c project_file3.c

# define the C object files
#
# This uses Suffix Replacement within a macro:
#   $(name:string1=string2)
#         For each word in 'name' replace 'string1' with 'string2'
# Below we are replacing the suffix .c of all words in the macro SRCS
# with the .o suffix
#
OBJS = $(SRCS:.c=.o)

# define the executable file
MAIN =
```

Dissecting a Makefile: MAIN specifies the name of the build file (the binary executable)

```
CC = gcc
# define any compile-time flags
CFLAGS =

# define any directories containing header files other than /usr/include
INCLUDES =

# define library paths in addition to /usr/lib
#   if I wanted to include libraries not in /usr/lib I'd specify their path using -Lpath
LFLAGS =

# define any libraries to link into executable:
LIBS =

# define the C source files
SRCS =  project_file1.c project_file2.c project_file3.c

# define the C object files
#
# This uses Suffix Replacement within a macro:
#   $(name:string1=string2)
#         For each word in 'name' replace 'string1' with 'string2'
# Below we are replacing the suffix .c of all words in the macro SRCS
# with the .o suffix
#
OBJS = $(SRCS:.c=.o)

# define the executable file
MAIN = my_project
```

Dissecting a Makefile (continued)

```
#
# The following part of the makefile is generic; it can be used to
# build any executable just by changing the definitions above and by
# deleting dependencies appended to the file from 'make depend'
#

.PHONY: depend clean

$(MAIN): $(OBJS)
	$(CC) $(CFLAGS) $(INCLUDES) -o $(MAIN) $(OBJS) $(LFLAGS) $(LIBS)

# this is a suffix replacement rule for building .o's from .c's
# it uses automatic variables $<: the name of the prerequisite of
# the rule(a .c file) and $@: the name of the target of the rule (a .o file)
# (see the gnu make manual section about automatic variables)
.c.o:
	$(CC) $(CFLAGS) $(INCLUDES) -c $<  -o $@

clean:
	$(RM) *.o *~ $(MAIN)

depend: $(SRCS)
	makedepend $(INCLUDES) $^

# DO NOT DELETE THIS LINE -- make depend needs it
```

How to build with make

- To invoke a Makefile and build a project simply type “make” on the command line in the same directory as the Makefile.

The motivation for cmake

- Problem: We live in a world with different operating systems (Unix, Linux, BSD, Microsoft's Windows 10, Windows 11, Apple's iOS, etc.).
  - Each of these operating systems locates essential system files in different locations (e.g. compilers, linkers, shared libraries).
  - How can one distribute source code to all these different operating systems and yet ensure that source code can be built for these systems? The answer: cmake!

- Cmake is a more powerful variant of make.
  - With cmake, it is possible to search and identify where on a client's computer a series of required dependency files (libraries, projects, etc.) are located, regardless of the operating system!
  - Cmake can then create a Makefile that is custom-crafted for the client's specific computer and dependent file versions.
  - Ultimately, the client can then build an application that is specific to their operating system.

How to build with cmake

- Building a project with cmake is usually quite simple and involves two commands:
- `cmake .`
- `make`

For further reading

- *Managing Projects with GNU Make* (3rd Edition) by Robert Mecklenburg. ISBN: 9780596006105