Bruce Bolden
Bruce Bolden
February 12, 2015

Lab Assignment #5
10 Points
Due: February 26, 2015

1 Make

Using the command line to build programs consisting of multiple files can be error-prone, not to mention tedious. Writing shell scripts can alleviate some of the problems, but may require some files to be compiled even though they do not need to be compiled.

Consider the process to build the stack example discussed in lecture. The executable depends on three source (`.cpp`) files and two header (`.h`) files.

The program can be compiled using:

```
g++ testStack.cpp  link.cpp  stack.cpp
```

Recall that `link.cpp` depends on `link.h`, `stack.cpp` depends on `link.h` and `stack.h`, and `testStack.cpp` depends on `stack.h`. This means that if `link.h` is changed, both `link.cpp` and `stack.cpp` must be recompiled. Similarly, if `stack.h` is changed, both `testStack.cpp` and `stack.cpp` must be recompiled. Overwhelmed with little details? Probably not, but this is a small test program, think about the process to build a larger application that contains hundreds of files.

One of the most useful development tools under Unix is `make` (there are several different versions to choose from, e.g., CMake, GNU Make, iMake, etc.). All versions of `make` use rules defined in a `makefile` to perform the desired actions (these actions can be very complicated).

1.1 Make files

The *rules* in a make file tells `make` how to execute commands to build a *target* file from *source* files. It also specifies a list of dependencies of the target file.

Makefiles can contain comments. Comments start with a `#` and are used to describe what is happening in the makefile or to hide definitions from `make`.

Sample makefile for the stack program:

```
#  Makefile
#    a makefile for the stack example.

#  SHELL =  /bin/sh

#  CPP = /lib/cpp $(STD_CPP_DEFINES)
    CXX = g++

#   CCOPTIONS =
#      CFLAGS = $(CDEBUGFLAGS) $(CCOPTIONS) $(ALLDEFINES)
#      RM_CMD = $(RM) *.o core

SRCS= teststack.cpp \
    stack.cpp \   #  Tab starts line!
    link.cpp

OBJS= teststack.o \
    stack.o \
    link.o

teststack.o: stack.h

stack.o: stack.h \
    link.h

link.o: link.h

PROGRAMS = testStack

all::  $(PROGRAMS)

testStack: $(OBJS)
    $(CXX) -o $@ $(OBJS)

clean::
    $(RM) testStack

latex::    #  typeset notes
    latex stack.tex

############################################################
# common rules for all Makefiles - do not edit

emptyrule::
```

```
clean::
    rm *.o
```

1.2 Other `makefile` Names

Makefiles can be defined in files named something other than `Makefile` or `makefile`. To use them, the command line option `-f` must be used. For example:

```
% make -f app.make
```

The `-f app.make` option to `make` tells `make` to read the rules defined in the file named `app.make`.

1.3 Experiments

1. Copy the makefile `Makefile` and the stack-related source files (`.cpp` and `.h`) from:
   http://www.cs.uidaho.edu/~bruceb/cs121/Labs/Make

2. Modify (or `touch`) one of the header files then invoke `make testStack`.

   - What files are compiled? (surprised?)
   - What is the last argument?

3. Using the shell, type `ls` (*return*), then type `make clean`, then type `ls` again. What happened?

4. Make a copy of the makefile for building the stack program, then revise the makefile to build the Queue or Binary Search Tree code (these programs are located on the *Sample Programs* web page) that we will be discussing in class.

1.4 Deliverables

Annotate a `script` session to demonstrate:

1. Answers to the three questions above. They end in question marks.
2. Contents of the modified Makefile.
3. Results of cleaning the directory (hint: before and after).
4. Proof that you built the Queue or Binary Search Tree program:

   Output of `make` **and** output of the resulting test program

1.5 References

Introduction to Make, Jennifer Vesperman
http://www.linuxdevcenter.com/pub/a/linux/2002/01/31/make_intro.html

O'Reilly, *Managing Projects with make*, 2nd Edition, 1991
http://shop.oreilly.com/product/9780937175903.do

GNU Make
http://www.gnu.org/software/make/

Makefile conventions
http://www.gnu.org/prep/standards/html_node/Makefile-Conventions.html#Makefile-Conventions

CMake
http://www.cmake.org/