COSC 3750

GCC, Make, GDB

Kim Buckner

University of Wyoming

Feb 02, 2023

Documentation

- Both **make** and **gcc** have man pages.
- These are probably good enough but ...
- It might be easier finding the info you want in the online GNU documentation.
- For gcc (on the department machines) check out gcc.gnu.org/onlinedocs/gcc-8.5.0/gcc/

GCC

- The GNU Compiler Collection
- Replacement for vendor specific versions of cc, the C compiler
- Also does C++, Fortran (g77 and gfortran), Java, Objective C and more.
- Available on almost all versions of Linux and UNIX.
- May have to install explicitly.

gcc

- The compiler actually is a set of programs
- We are going to discuss the C/C++ compiler
- The compilation process is:
    - preprocess the input files if necessary,
    - generate assembly language from the preprocessed input,
    - run the assembler on the compiler output to generate object code,
    - and link the object object code to create an executable.

The Preprocessor

The preprocessor

- Called *cpp*, the **C p**re**p**rocessor.
- This is a fairly simple program that just processes directives.
- The directives are of the form #DIRECTIVE_NAME
- Some of them take arguments and some do not.
- These directives allow you to more easily control the compilation.

#include

- The required argument is a name (of a file) enclosed in delimiters.
- If the name is enclosed in < > then file is searched for in the list of standard directories.
- If the name is enclosed in " " then the file is FIRST searched for in the directory containing the current file.
- `cpp -v /dev/null -o /dev/null`

Changing the path

- The -I option for gcc is passed to cpp to specify a list of “include” directories that are searched BEFORE the standard ones.
- Handy if you have your own directories for header files.

#define

- Simply defines a macro (name)
- Can assign values or strings.
- Very simplistic BUT also handy.
- Many uses, such as keeping files from being included multiple times, conditional compilation, etc.

Testing

- #if, #ifdef, #ifndef
- #if tests an arithmetic expression
- The other two test whether or not macros have been defined.
- Defined is taken to mean “value other than 0”.
- Also #else and #elif
- #endif is required.

More on directives

- If you want/need more information there are two sources.
- Readily available is the info page

```
info cpp
```

- Or the GNU manual on cpp at gcc.gnu.org/onlinedocs/cpp/.

The Compiler

Compilation

- There are a huge number of options.
- Will only cover the ones I think most important.
- The first is the output file name, -o *filename*
- By default, the executable gcc creates is *a.out*.

Why?

- Actually, in this case the Wikipedia page is probably the best resource.
- Someone put in a huge amount of effort to find all the information.
- They may have made some mistakes, but after checking out the references I am satisfied with the explanation.
- “[a]ssembler [out]put”

Debugging

- In order to use the debugger effectively, have to have the information in the output.
- Turn on with -ggdb.
- Will see other references to just -g.
- I find that -g does not always put in enough/correct information for gdb.

Warnings

- By default, some warnings are printed and all errors.
- Good programmers become familiar with ALL the warnings and **fix** them
- The simplest way to get them is -Wall.
- It does not really give all warnings but all really useful ones.

Output type

- Can stop the compilation process part way, sometimes useful
- Only generate object code, -c
- Do not assemble, but generate the assembly language, -S
- Preprocess but do not compile, -E

Linking

- Normally just done.
- May need to specify additional libraries to link in.
- Like the include path there is a library path.
- Add directories with the -L option
- Specify the libraries with the -l*libname* option

'man sqrt'

Make

Make

- This manages files
- Huge number of options and rules. Read the GNU Make Manual www.gnu.org/software/make/manual/
- Basics
    - Variables
    - Targets
    - Prerequisites
    - Recipe

Variables

- Declared like *sh* variables.
- There are a number predeclared.
- Can access the environment variables.
- Be safe, make sure you have all variables declared correctly.

(more . . .)

- Must precede the variable name with a $
- BUT must enclose the name in () or {}, otherwise assumed to be a single character name.
- Generally, by convention, we use all caps for variable names.
- Do NOT have to use the variables, just makes it neater and more convenient

Targets

- Several ways to specify. Will discuss the simplest.

  target: *prerequisites*

      recipe
- The target can be anything but is generally the file you want to “make”.

Decide what to do

- If the target does not exist, make it.
- If any prerequisite is newer than the the target, make it.
- Of course have to ensure that all prerequisites are up to date.
- If they exist and correspond to a target, verify they are up to date.
- If no rule and they exist, assume they are up to date.

Recipes

- The recipes are just a set of shell commands
- They MUST be preceded by a tab. Not 8 spaces, but a tab character.
- If you copy with the mouse, probably will get spaces.
- Can be any number of commands
- If the lines are too long, escape the newline

(more . . .)

- Each line is a separate command.
- If any one “fails”, MAKE exits.
- By default, it echoes the line before executing it.