CSSE2310 Teaching Resource

Tutorials

Makefiles

Contents

Made with ❤ by the CSSE2310 teaching staff

Template sourced from Joel Fenwick and Justin Goldizen

September 9, 2019

Intro

This resource is intended to give you a quick run down of Makefiles which are a build tool that will be mandatory in this course. It is used for your assignment submissions and saves a lot of developer time.

0.1 Pre Reqs

- Basic knowledge of linux is needed, please make sure you have completed the (*Linux Intro*) available on blackboard under "Learning Resources".
- Up to chapter two in the (*ctute*), please finish at least those sections before trying the tutorial.

1 About

Makefiles are small configuration files which can be used by the program `make` to build large projects or other complicated procedures. The usefulness comes from being able to chain large sets of commands into a single action which can be executed on the command line.

1.1 Getting Started

To begin working with a Makefile we need to create an empty file with the name `Makefile` in a clean directory. Next we want to create a basic **Rule**, which can be copied from below. Please note that on line 1 it is indented using tabs not spaces. Tabs are mandatory for **Rules**.

```
0 bob: bob.c
1 	gcc -Wall -pedantic -std=c99 bob.c -o bob
```

A **Rule** is the basic building block of a makefile, it's made up of 3 different parts. The first on line 0 before the colon is the name of the rule, which is known as the target. The target for the example above is "bob" because this rule will compile the bob program.

On line 0 are the dependencies, these can be files on the system or other targets which we will see later with a special target called "all". Make uses these dependencies to decide if it should rebuild the target. It does this by seeing if the dependencies are newer than the target, in our case the program named bob.

Lastly, on line 1 we have the command/commands that will be run when this target is triggered to run. Multiple commands can be on a new line underneath each other as long as they are indented by a singular tab and not by spaces.

NOTE

In vim you can enable showing whitespace characters like tabs with `:set list` to enable and `:set list!` to disable.

1.2 Other Targets

Not all targets need to generate output and some are used to change settings or clean the workspace. We will now work to build a target called "all" which its only purpose is to build other targets.

To build the all target we know that we wont be producing any output called "all" we just need a target that runs every time we ask for it to be run. This is done by making the target a "phony target". We also know that targets can have other targets as dependencies so we list bob as a dependency of the "all" target.

```
0 .PHONY: all
1
2 all: bob
3
4 # bob creates the bob program
5 bob: bob.c
6 	gcc -Wall -pedantic -std=c99 bob.c -o bob
```

line 0 shows how to make a target phony by using the special target name `.Phony` and listing all the other targets as dependencies that we want to be phony. Line 2 shows the all target that has the bob target as a dependency and no commands. If you were to run `make all` it would run the all target and then trigger bob to run if it needs to be rebuilt.

1.3 Variables

Suppose we have a much larger program that builds itself into small linkable files and then combines them. This would require different targets all with similar compilation options like "-Wall". But say for example that we wanted all these *targets* to also use "-pedantic" as well, we would need to edit every *command*. Instead with makefiles, we can use variables so that we only need to edit the variable and it changes every command that uses it.

With the example bob makefile given above, add the following line above the target.

```
CFLAGS = -Wall
```

This defines a variable that we can use within our commands by replacing `-Wall` with `$(CFLAGS)` in our command line.

```
bob: bob.c
	gcc $(CFLAGS) bob.c -o bob
```

Variables are also able to be changed during the lifetime of the program, say for example that we wanted to add more compilers warnings if we run make with `make extra bob`. This could be accomplished by having the following:

```
0  # Variables
1  CFLAGS = -Wall -pedantic -std=c99
2  # Targets that do not generate output
3  .PHONY: extra
4  # Set the default target if you run make with no parameters
5  .DEFAULT: bob
6
7  # extra adds more compiler warnings, like the unreachable-code and default case
8  #   warnings.
9  extra: CFLAGS += -Wswitch-default -Wunreachable-code
10
11 # bob creates the bob program
12 bob: bob.c
13 	gcc $(CFLAGS) bob.c -o bob
```

2 Exercises

2.1 Setup

1. Connect to MOSS or a local Linux installation. (*linuxintro chapter 1*)
2. Create a new directory to start working in. (*linuxintro chapter 3*)
3. Extract `~s4356897/public/2310/makefiles/make.ex.tar.gz` (*linuxintro chapter 8*)

2.2 Ex1

Your task is to accomplish the following:

1. Write a Makefile to build "hi" from hw.c

   NOTE

   What happens when your run this makefile multiple times?

2. Modify the Makefile to compile and link separately.
3. Add a "clean" target.
4. Modify the Makefile to use a variable to specify compile flags and then build with debug.

2.3 Ex2

Your task is to write a Makefile to build the program called "thromborax" from the supplied files, be sure to build each `.o` separately.

3 Additional Resources

We have only touched on the surface of Makefiles; there are many other tricks and advanced features but they are not within the scope of the course. Feel free to learn in your own time as it may help with future courses.

Another resource written by a long time teaching staff member Joel Addision can be found here.
https://uni.joeladdison.com/csse2310/programming/makefiles
The link above also contains other useful pieces of information for CSSE2310.

The GNU make manual is available here: https://www.gnu.org/software/make/manual/make.html

4 Feedback

If you wish to leave feedback for this tutorial, feel free to leave a comment on piazza, search for the "Tutorial Feedback" thread.