Makefiles

Kyle Corry

October 8, 2017

Contents

1 What is a Makefile?

It is a way to make building things easier. Mainly used for C, but it can be used for anything. It is used to build things that depend on other things and making that easier.

frosted_cake example A frosted cake depends on frosting and cake. Frosting depends on butter, cream, and sugar. Cake depends on sugar (shared dependency), flour, milk, and cracked_eggs. Cracked_eggs depend on eggs. Dependencies are like directed acyclic graphs - this means that dependencies don't depend on things that depend on them.

The makefile for this would be:

```
frosted_cake: frosting cake
        spread frosting onto cake

frosting: butter cream sugar
        ...

cake: sugar flour milk
        ...
```

You don't need to specify how to make the lowest level dependencies. You need to specify how to make things that have dependencies.

2 A C Program

main.c uses things in stuff.c and things.c. Both stuff.c and things.c use utils.c. This means the executable depends on 4 .o (object) files: main.o, stuff.o, things.o, utils.o. main.o depends on main.c, stuff.h, and things.h. Stuff.o depends on stuff.h, stuff.c, and utils.h. Things.o depends on things.h, things.c, and utils.h. Utils.o needs utils.h and utils.c.

An object file needs:

1. .c file
2. .h file
3. .h files of part of the program it references.

The C example would have the following makefile:

```
executable: *.o
        gcc $^ -o executable

main.o: main.c stuff.h things.h
        gcc main.c -c

...
```

The $^refers to all dependencies. $? refers to dependencies that have changed since last build. The $@ is the target. $x refers to the variable x.

If you want something to be configurable, make it a variable:

ex. $PROGRAM\_NAME = project$
Then go through and insert the variable:

```
PROGRAM_NAME=project

$(PROGRAM_NAME): *.o
        gcc $^ -o $(PROGRAM_NAME)
```

When something changes, everything that depends on it will need to be recompiled until the last item which nothing depends on is recompiled. For example, if utils.c changes, utils.o changes, and then executable is rebuilt. So make will only recompile things that need to be changed. It knows what needs to be rebuilt based on the modified date.

2.1 Generics

The following code will make the object file for all c files:

```
%.o: %.c *.h
        gcc $> -c
```

This is useful for things other than C. Basically anything with dependencies.

2.2 Invoking Make

make target_name will build the make file. It is also convention to add a make recipe called all that has the final executable name on the right. It is also convention to have a clean recipe, it should delete all object files and the executable.

```
clean:
        rm -f *.o executable
```

You can also have a make run. This will just execute the executable. Make docs will also make the docs of the program.

```
run:
        ./executable
```

The make file is generally named *makefile* or *Makefile*. There is also a command called all, which will trigger the executable building command. It has the following format:

```
all: executable
```