
Programmeren C 2010

GNU Make: An introduction

assistent(s) Tim van Deurzen, Merlijn Wajer
email programmeren.java@gmail.com

author(s) Tim van Deurzen

1 Introduction

GNU Make is one of the most useful and important tools you will encounter when programming in C. You've already encountered 'make' when you were making the zero'th and first assignments. There, a file was provided called a 'Makefile'.

In this file the rules for compiling your program are defined. Here is a very simple 'Makefile' for compiling a program called 'assign1.c' and creating an executable called 'main'.

```
1      # Set the C compiler.
2      CC = gcc
3
4      # Set the flags for the C compiler.
5      CFLAGS = -Wall -std=c99 -pedantic
6
7      # Macro for compiling.
8      COMPILE = $(CC) $(CFLAGS) -c
9
10     # Macro for the output of compilation.
11     OUTPUT = -o $@
12
13     # Rule for creating an object file by compiling a c file.
14     # '%' is a wildcard meaning any file with the suffix '.c' will be
15     # compiled.
16     %.o: %.c
17         $(COMPILE) $< $(OUTPUT)
18
19     # Rule for building an executable called 'main' using assign1.o
20     main: assign1.o
21         $(CC) $(CFLAGS) $(OUTPUT) $^
```

Listing 1: A simple Makefile.

Let's look at this file line by line. In the first line the variable *CC* is defined and set to 'gcc'. This way *make* knows we want to use the *Gnu C Compiler*. The second assignment is *CFLAGS*. This macro defines the flags that will be passed to gcc.

Now we come to a macro that is not necessary but makes the file just that bit more readable. The macro *COMPILE* allows calling the C compiler with the selected flags. It also tells the compiler we only want to compile the files and not build an executable yet. This is not that important for project with one file, but very important when more than one file is to be compiled and linked together to get one executable.

The next macro is also just for readability. Do note, however, the '\$@' that is used. This is an 'Automatic Variable' and represents *the target*. The target is the first part of a rule ('main', '%.o').

Now we come to the first *rule* and this looks a bit more complex than the previous lines. This rule defines what should happen when an arbitrary object file is needed. The '%.o' allows this rule to be used to build any object file from any ('%.c') C source file. The next line after this rule is indented by one or more *TABS*. Indenting in Makefiles should always use tabs and not spaces. On this next line is defined what should be done with the '%.c' files to create

the required object files. In this case we want to call our COMPILE macro and feed it the first prerequisite ('\$<'). Prerequisites are the files names after the colon ('%.c', 'assign1.o'). The output file will get the name of the target '%.o'. The % will be replaced by the filename of the file being compiled.

Finally we have the most important rule, the one defining:

- what is needed to build the executable (prerequisites): **assign.o**
- what should be done with these prerequisites: **call the C compiler with the object files and create an executable.**
- and, what the executable should be called (the target): **main**

In this case only one object file is used to create an executable, but it is possible to combine several object files into one executable. This is achieved by using another *automatic variable*: \$^ which grabs all the prerequisite files. The nice thing about make is that it checks which files have been altered before starting to compile everything. This is especially nice for project with a lot of files. If one file is changed only this file is re-compiled and then the executable is built directly. Make also checks if all the prerequisites are available and if not it uses the rule for that prerequisite to build it. Without a rule the prerequisite will not be built and make will exit with an error. For this reason it is nice to have the rule on line 16. This automatically compiles all C source files when an object file is listed as prerequisite for any rule.

As a computer scientist, or a programming enthusiast you should definitely have a look at Gnu Make and learn how to write your own Makefiles. It's a skill you'll long cherish!

Check out the tutorial here: <http://www.gnu.org/software/make/manual/make.html>