HowTo Write Simple Makefiles

C++ (http://www .puxan.com/web/tag/cplusplus/) Open source (http://www .puxan.com/web/tag/open-source/)

What are Makefiles?

Make is a GNU tool to build executables. The main advantage of using this tool is that it only recompiles the needed parts of the project. Therefore if you are working with large projects you won't waste your time recompiling sources that haven't been modified. Besides you can compile using one single command: **make**.

Makefiles define how the project have to be build and which are the source code dependencies. Using this dependencies and checking the modification time Make automatically determines which parts have to be recompiled.

Rules

Makefiles contain a set of rules that indicate what have to be done and how. The first rule will be the one used by default when typing **make**. A part from that, the order of the rules in the file is not important. You should also know that the actions that have to be taken to achieve a target have to be preceded by a tabulator and no other characters (nor spaces).

Rules follow the schema:

```
Rule: Dependencies
        Actions
```

Lets see an example. In this piece of a Makefile, to get the object (example.o) we need the source code (example.cpp). The second line indicates that the way to get the object is using the gcc command. We could also add header files in the dependencies.

```
example.o: example.cpp
        gcc -c example.cpp
```

If you want to achieve one specific rule you can type **make** followed by the name of the rule.

Variables

As always, variables help us avoiding rewriting the same thing many times and to easily change some of the parameters we want to use. Lets see how to use them with a very small example where a variable called CC will hold the complier command.

```
# Variable declaration
CC = gcc

example.o: example.cpp
        $(CC) -c example.cpp
```

Simple example

Lets see a simple full example of a Makefile. In this case the first target is **example**, so when we run **make** this target will be the one to achieve. This target has a dependency of **example.o**, so before executing the **gcc** command it tries to satisfy the dependency. To do that, **make** checks if a file with this name exists, if it doesn't it looks into the Makefile for a target with this name. If a target is found, this one have to be achieved before going back to the previous target, and so on.

```
# Declaration of variables
CC = g++
CC_FLAGS = -w

# Main target
example: example.o
        $(CC) example.o -o example

# To obtain example.o
example.o: example.cpp example.hh
        $(CC) $(CC_FLAGS) -c example.cpp

# To remove generated files
clean:
        rm -f example example.o
```

See also

HowTo Write Generic Makefiles (/web/howto-write-generic-makefiles/)

References

GNU Make user (http://www.gnu.org/software/make/manual/make.html)