Automatically generate makefile dependencies

Content

Tested on

Debian (Etch, Lenny, Squeeze)

Ubuntu (Lucid, Maverick, Natty, Precise, Trusty)

Objective

To automatically list dependencies between the files in a body of source code, in a form suitable for use within a makefile

Background

When `make` builds a program it uses timestamps to determine which files need to be rebuilt and which do not. In order to do this it needs to know how the source, intermediate and executable files depend on each other. That information is provided by the makefile in the form of rules. For example, the line:

```
main.o: main.c foo.h bar.h
```

indicates that the object file `main.o` depends on the source file `main.c` and on the header files `foo.h` and `bar.h`.

Creating these rules manually is both tedious and error-prone. Mistakes can be difficult to diagnose, because they result in a compiled program that does not correspond to the source code. For these reasons, the use of an automated mechanism for identifying dependencies is highly desirable for all but the simplest of programs.

Scenario

Suppose you have a makefile which compiles all files with the extension `.c` that are located in the same directory, then links them into an executable called `hello_world`:

```
SRC = $(wildcard *.c)

hello_world: $(SRC:%.c=%.o)
        $(LD) -o $@ $^
```

This will correctly rebuild the executable when one of the `.c` files is changed, but not when an `.h` file included by one of the `.c` files is changed.

You wish to rectify this deficiency without having to explicitly specify which headers are used by which source files. You are building the program using GCC and GNU Make.

Method

GCC has the ability to create lists of dependencies as a program is being compiled. This feature is invoked using:

- the `-MD` option if you wish to include system headers, or
- the `-MMD` option if you do not.

Including system headers is the more robust alternative, however they change rarely and can greatly increase the number of dependencies listed.

By default the dependency list is written to a file with the extension `.d` and is in the correct format for inclusion in a makefile. All you need do to automate the process is:

- arrange for `-MD` to be added to the list of options that are passed by `make` to the compiler, then
- incorporate the dependency files into the makefile by means of an `include` directive.

A useful embellishment is to invoke the `-MP` option, which creates an empty rule for generating each dependency. This suppresses the errors that would otherwise occur when a header file is deleted or renamed.

Because `-MD` and `-MP` are preprocessor options, the standard method for setting them within a makefile is to append them to `CPPFLAGS`:

```
CPPFLAGS += -MD -MP
```

The include directive must list the dependency files. Most makefiles will already have a list of source or object files, from which the names of the dependency files can be obtained by substitution. In this particular example it is possible to use the variable `SRC` which is a list of source files:

```
-include $(SRC:%.c=%.d)
```

The purpose of the initial hyphen is to suppress the error messages that would otherwise appear when the dependency files do not already exist.

The include directive must not precede the default target, otherwise the first rule to be included would supercede the default target. A simple way to prevent this from happening is to place the include directive at the end of the makefile.

Do not add a rule for explicitly building the dependency files. The method presented here will build or rebuild those files when needed without them being an explicit part of the dependency graph, so adding a rule would merely create unnecessary work for `make`. The way this is achieved is subtle but effective:

- If a dependency file needs to be updated then so does the corresponding object file.

- A side-effect of updating an object file is to update the corresponding dependency file.
- Therefore, if the object files are updated when necessary then so are the dependency files.

Here the complete makefile for the scenario described:

```
CPPFLAGS += -MD -MP

SRC = $(wildcard *.c)

hello_world: $(SRC:%.c=%.o)
        $(LD) -o $@ $^

-include $(SRC:%.c=%.d)
```

Alternatives

Use the -M option with sed

The GNU Make manual recommends creating the dependency files explicitly using a pattern rule:

```
%.d: %.c
        @set -e; rm -f $@; \
        $(CC) -M $(CPPFLAGS) $< > $@.$$$$; \
        sed 's,\($*\)\.o[ :]*,\1.o $@ : ,g' < $@.$$$$ > $@; \
        rm -f $@.$$$$
```

This has the advantage of using the `-M` option in place of `-MD`, the former being more widely supported by compilers other than GCC. Disadvantages are that:

- the source code must be preprocessed twice (once to create the dependency files and once to compile the object code), and
- this method does not gracefully handle dependencies that have been deleted or renamed (the relevant dependency file must be manually deleted).

Use GNU Automake

Automake is a tool for automatically generating portable makefiles that conform to the GNU Coding Standards. It is normally used in conjunction with Autoconf, and often with Libtool. It generates dependencies using the compiler if it can, or using `makedepend` if not.

Tags: make