Introduction to UNIX "make"

The Unix utility **make** is a tool that automates the building of executable programs from one or more source files. **make** uses a *description* file (called a *makefile*) supplied by the programmer to determine which commands need to be executed in order to produce the executable program requested. The description file contains the names of *targets*, for example executable files or object files, along with a list of *dependencies* for each target. If any file in the list of dependencies is newer than the target file, a list of command(s) given on the following line(s) are executed. These command(s) presumably will produce the target file by compiling and/or linking the files in the dependencies list. Files that appear in a dependency list may also be a target elsewhere in the description file. Commonly an object file will be in a dependency list for an executable file and also will be found as a target with one or more source files in its dependency list.

Once you have a file that describes your program, you can build the program at any time by entering the command **make**. **make** looks in the current directory for a file named **makefile** (or, if no such file, it looks for a file named **Makefile**). On the **make** command-line, a specific target to build can be specified as an argument. If no target is specified on the command-line, **make** attempts to build the first target defined in **makefile**.

For each target in the description file, information describing that target is given in the following format:

target **:** *dependency-list*
action(s)

The *target* is the name of a file. Targets can be executable files, object files, or program text files. The *dependency-list* is a (possibly empty) list of prerequisite files used in some way to create the target. The *action(s)* are one or more Unix commands that **make** uses to construct or update the target. The actions are typically compile or link commands. If any of the prerequisite files in the dependency-list are newer than the target, or the target does not exist, the action is executed. The target *must* start in column 1. The action(s) must be on the next line (no blank lines between it and the target/dependencies line) and the action line(s) *must* start with a **tab** character. Multiple commands can be executed to make a single target by giving each command on a separate line. A blank line signals the end of the description for the current target.

Example

Consider a program that consists of a main source file, `ola1.cpp`, two auxiliary files, `class1.cpp` and `class2.cpp`, and their corresponding header files, `class1.h` and `class2.h`. This system could be described by the following file:

```
# File "makefile" used to build "olaX" executable.
# Lines beginning with # are comments.
#

olaX: olaX.o class1.o class2.o
        c++ olaX.o class1.o class2.o -o olaX

# olaX.cpp must be recompiled if it or the class1.h header file have changed
olaX.o: olaX.cpp class1.h
        c++ -c olaX.cpp

# recompile class1.cpp if either class1.cpp, class1.h, or class2.h is
#   newer than class1.o.  Recall that the "-c" option means compile-only,
#   thereby creating a .o object file, but do not run the linker.
class1.o: class1.cpp class1.h class2.h
        c++ -c class1.cpp

# recompile class2.cpp if either class2.cpp or class2.h is newer than class2.o
class2.o: class2.cpp class2.h
        c++ -c class2.cpp
```

In the above example, program `olaX.cpp` directly uses some class(es) or function(s) declared in **class1.h** and so must be recompiled whenever **class1.h** has been changed. The **class1.cpp** implementation uses some class(es) or functions(s) declared in **class2.h**, as well as its own header file, and so must be recompiled whenever **class1.h** or **class2.h** change. Of course, if a source file itself changes, it must be recompiled.

A "`#`" in a line of a makefile starts a comment; it and the rest of the line are ignored. The local comments above are included to help explain the example; in practice such comments are usually omitted because the entries are self-explanatory to someone familiar with writing makefiles. Global comments, describing what the makefile is for, are always welcome.

For more information, see the man page for **make** (`man make`).