In the next three lectures, we will cover a tool that is used to manage the construction of files from other files. The most common example of this phenomenon is the process of compilation for programming languages: you have a set of source files, you want to compile each of them into object files, and finally you want to link those object files together, possibly with some library files, into an executable program.

The tool, called *make*, reads in a file containing a description of the dependencies between the files, and commands describing how to construct target files from dependent files. The *make* tool then automatically figures out the best way to construct the new files. Among other things, if some of the target files already exist and are still valid (that is, if there has been no modification to the dependent files), then these files don't need to be rebuilt.

Unfortunately, *make* suffers from the same problems as shells: there are many versions out there, all slightly incompatible. For the sake of discussion, we will concentrate on the GNU version of make, available on *babbage* as */usr/local/gnu/bin/make*. Much of what you will learn this week applies to other *make* tools, but some of the features or the syntax may be slightly different.

A sample makefile

The files that *make* reads and that contain the dependencies between files are called makefiles, and one typically stores the under the name *makefile*. Here is a sample makefile:

```
myapp : file1.o file2.o
      gcc -o myapp file1.o file2.o

file1.o : file1.c macros.h
      gcc -c file1.c

file2.o : file2.c macros.h
      gcc -c file2.c
```

This makefile describes the dependencies requires to compile the C source files *file1.c*, *file2.c* (and header file *macros.h*) into the executable called *myapp*. This makefile contains three rules, each rule describing a particular dependency. A rule is of the form:

```
target : dep dep dep
```

```
	cmd
	...
	cmd
```

Such a rule states that to build the file *target*, you first check if *target* does not exist, or if it is older than any of the dependency files *dep*. (Here, older is taken to with respect to the time of last modification, not the time of creation.) If any of these conditions hold, then the file *target* needs to be rebuild. To rebuild it, the commands specified after the rule are executed. It is very important to note that each line containing a command must start with a TAB character (i.e. press TAB at the beginning of the line). If this is not the case, you'll either get an error, or at best unpredictable behavior.

Consider the previous example. It states that you need to rebuild the file *myapp* if either it does not exist or if it's older than either *file1.o* or *file2.o*. To rebuild *myapp*, you invoke *gcc*, the C compiler, with the appropriate flags. Similarly for rebuilding *file1.o* and *file2.o* from *file1.c* and *file2.c*.

To "execute" such a makefile, you simply call *make* in the directory of the makefile. If the makefile is indeed called *makefile* (or *Makefile*), *make* will automatically find it and figure out if there are any files that need to be rebuild and if so rebuilds them. If the makefile is stored under another name, you can still use *make*, with an option: *make -f makefilename*.