Advanced Unix Tools	**Lecture 9**
CS214 Spring 2003	**Friday, March 7, 2003**

As you have noticed in the examples of makefiles I have been giving in class, many rules are essentially the same except for the name of the target and dependencies. For instance, to compile a C file, a rule typically looks like:

```
file.o : file.c
    gcc -c file.c
```

(potentially with additional flags, or with the use of variables to abstract away from the compiler, etc.) If there are many C files part of the project, each will have a rule of the above form. What you would like, of course, is a way to say something like: for every target with a *.o* extension, it depends on the corresponding file with *.c* extension, and to build it, you invoke such and such. That's what static pattern rules are for. A static pattern rule has the following form:

```
targets : patterntarget : patterndep dep ...
   cmd
   cmd
   ...
```

Intuitively, such a rule says: for all the targets *targets*, if it matches *patterntarget*, then it depends on *patterndep* and possibly *dep* and other fixed dependencies, and to build it you execute the following commands.

A typical static pattern rule for compiling a C file would look like:

```
objects = file1.o file2.o file3.o

$(objects) : %.o : %.c
    gcc -c $<
```

The pattern for the target, *%.o*, contains a *%* which intuitively matches any number of characters. If the target is *foo.o*, then *%* matches *foo*. Whatever matched *%*, called the *stem*, is substituted in the pattern for the dependency to establish the dependencies for that particular pattern. Hence, the above rule is equivalent to rules for compiling *file1.o* depending on *file1.c*, *file2.o* depending on *file2.c*, and *file3.o* depending on *file3.c*. The command uses a special makefile variable *$<*, which is automatically expanded by *make* inside a command line into the first dependency file. In the case above, it will be expanded into either *file1.c*, *file2.c*, or *file3.c*, depending on the actual dependency

file, which depends on the actual target. There are a few such special variables that you can use in command lines in rules; here's a partial list:

$@	current target
$<	first dependency file
$^	all dependency files
$*	stem
$?	all dependency files that are newer than target