**4.6 Phony Targets**

A phony target is one that is not really the name of a file; rather it is just a name for a recipe to be executed when you make an explicit request. There are two reasons to use a phony target: to avoid a conflict with a file of the same name, and to improve performance.

If you write a rule whose recipe will not create the target file, the recipe will be executed every time the target comes up for remaking. Here is an example:

clean:

rm \*.o temp

Because the rm command does not create a file named clean, probably no such file will ever exist. Therefore, the rm command will be executed every time you say ‘make clean’.

In this example, the clean target will not work properly if a file named clean is ever created in this directory. Since it has no prerequisites, clean would always be considered up to date and its recipe would not be executed. To avoid this problem you can explicitly declare the target to be phony by making it a prerequisite of the special target .PHONY (see [Special Built-in Target Names](https://www.gnu.org/software/make/manual/make.html#Special-Targets)) as follows:

.PHONY: clean

clean:

rm \*.o temp

Once this is done, ‘make clean’ will run the recipe regardless of whether there is a file named clean.

Phony targets are also useful in conjunction with recursive invocations of make (see [Recursive Use of make](https://www.gnu.org/software/make/manual/make.html#Recursion)). In this situation the makefile will often contain a variable which lists a number of sub-directories to be built. A simplistic way to handle this is to define one rule with a recipe that loops over the sub-directories, like this:

SUBDIRS = foo bar baz

subdirs:

for dir in $(SUBDIRS); do \

$(MAKE) -C $$dir; \

done

There are problems with this method, however. First, any error detected in a sub-make is ignored by this rule, so it will continue to build the rest of the directories even when one fails. This can be overcome by adding shell commands to note the error and exit, but then it will do so even if make is invoked with the -k option, which is unfortunate. Second, and perhaps more importantly, you cannot take advantage of make’s ability to build targets in parallel (see [Parallel Execution](https://www.gnu.org/software/make/manual/make.html#Parallel)), since there is only one rule.

By declaring the sub-directories as .PHONY targets (you must do this as the sub-directory obviously always exists; otherwise it won’t be built) you can remove these problems:

SUBDIRS = foo bar baz

.PHONY: subdirs $(SUBDIRS)

subdirs: $(SUBDIRS)

$(SUBDIRS):

$(MAKE) -C $@

foo: baz

Here we’ve also declared that the foo sub-directory cannot be built until after the baz sub-directory is complete; this kind of relationship declaration is particularly important when attempting parallel builds.

The implicit rule search (see [Implicit Rules](https://www.gnu.org/software/make/manual/make.html#Implicit-Rules)) is skipped for .PHONY targets. This is why declaring a target as .PHONY is good for performance, even if you are not worried about the actual file existing.

A phony target should not be a prerequisite of a real target file; if it is, its recipe will be run every time make goes to update that file. As long as a phony target is never a prerequisite of a real target, the phony target recipe will be executed only when the phony target is a specified goal (see [Arguments to Specify the Goals](https://www.gnu.org/software/make/manual/make.html#Goals)).

Phony targets can have prerequisites. When one directory contains multiple programs, it is most convenient to describe all of the programs in one makefile ./Makefile. Since the target remade by default will be the first one in the makefile, it is common to make this a phony target named ‘all’ and give it, as prerequisites, all the individual programs. For example:

all : prog1 prog2 prog3

.PHONY : all

prog1 : prog1.o utils.o

cc -o prog1 prog1.o utils.o

prog2 : prog2.o

cc -o prog2 prog2.o

prog3 : prog3.o sort.o utils.o

cc -o prog3 prog3.o sort.o utils.o

Now you can say just ‘make’ to remake all three programs, or specify as arguments the ones to remake (as in ‘make prog1 prog3’). Phoniness is not inherited: the prerequisites of a phony target are not themselves phony, unless explicitly declared to be so.

When one phony target is a prerequisite of another, it serves as a subroutine of the other. For example, here ‘make cleanall’ will delete the object files, the difference files, and the file program:

.PHONY: cleanall cleanobj cleandiff

cleanall : cleanobj cleandiff

rm program

cleanobj :

rm \*.o

cleandiff :

rm \*.diff