reevaluate makefile variables

Asked 15 years, 10 months ago Modified 12 years, 4 months ago

Viewed 5k times

11

Is there a way to reevaluate a variable's definition upon each use? For example:

```
MAP_FILES = $(shell find $(TMP) -name "*.map")

all: generate_map_files work_with_map_files

generate_map_files:
   ./map-builder

work\_with\_map_files: $(MAP_FILES)
   ./map-user

%.map:
   ./map-edit $@
```

So, MAP_FILES will be evaluated when the makefile is read, and if there are no .map files in the directory $TMP the variable will be empty. However after the generate_map_files rule is completed there will be .map files in the directory and I would like the list of those .map files to be prerequisites to the work_with_map_files rule.

I don't know the filenames of the .map files before they are generated so I can not declare a variable with filenames explicitly. I need the variable to be set with the

list of map files once they have been generated. Any suggestions would be very helpful. Thanks.

variables makefile

Share

Improve this question

Follow

asked Feb 16, 2009 at 22:00

bob

3 Answers

Sorted by: Highest score (default)

5

You might try doing a recursive make, something like

```
MAP_FILES = $(shell find $(TMP) -name "*.map")

all: generate_map_files

generate_map_files:
   ./map-builder; $(MAKE) work_with_map_files

work\_with\_map_files: $(MAP_FILES)
   ./map-user

%.map:
   ./map-edit $@
```

Share Improve this answer

Follow

answered Feb 16, 2009 at 22:39

mbyrne215

2,334 ● 4 ● 21 ● 16

4

With GNU make, you can take advantage of the makefile remaking feature, which causes GNU make to automatically restart if any of the included makefiles are changed during the initial pass. For example:

```
.PHONY: map_files.d
-include map_files.d
map_files.d:
        ./map_builder
        echo "work_with_map_files: `ls *.map`" >
map_files.d

work_with_map_files:
        ./map_user
```

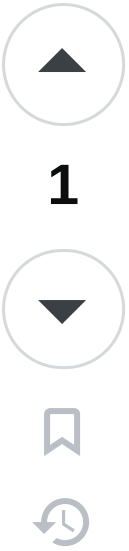
In addition to the makefile remaking feature, this solution makes use of the fact that GNU make allows you to specify multiple lines of prerequisites for a target. In this case, the map file prereqs are declared in the dynamically generated map_files.d file. Declaring map_files.d as a PHONY target ensures that it is always regenerated when you run the build; that may or may not be appropriate depending on your needs.

Share Improve this answer Follow

edited Jul 26, 2012 at 2:34

answered Mar 22, 2009 at 5:21

Eric Melski
16.8k 3 39 54



In general this is not possible in Makefiles, because to determine what targets to make and in what order, `make` needs to know their dependencies in advance before the rules are executed.

In your example, how would `make` know when to evaluate the `$(MAP_FILES)` in the `work_with_map_files` rule? The order is not explicitly defined, but deduced from the dependencies. In your example you want it to be evaluated after the `generate_map_files` rule was executed, but there is no way for `make` to know that because it needs the to know the value of this variable for the dependencies which are needed to determine the order at which this value would be evaluated - that is a self-referential loop.

One simple trick would of course be to run make twice - you can have that done automatically by adding a `make work_with_map_files` command after the `./mapbuilder` command in the `generate_map_files` template, but be careful with this in general because if `work_with_map_files` would actually be declared to depend on `generate_map_files` (which it should) this would lead to an infinite recursive `make` loop. And of course this defeats the idea of `make` automatically determining the order. Else you would need a `make` replacement which can be hinted on such orders and make multiple passes.

This is the reason that in bigger code bases with multiple include files, where one does not want to repeat the

include dependencies in the `Makefile`, `makedepend` is often used to generate a separate Makefile with those dependencies, which is included in the main `Makefile`. To build one then runs first `make depend` which calls `makedepend` to generate the include file dependencies, and then `make`.

Share Improve this answer Follow

edited Feb 16, 2009 at 22:49

answered Feb 16, 2009 at 22:13

Tom Alsberg
7,023 ● 3 ● 30 ● 14