NAME

mymake – Make requested targets with sanity checking and no unnecessary outputs.

SYNOPSIS

${HOME}/bin/mymake [target1 target2...]

DESCRIPTION

This shell script checks for a file named Makefile and stops if one is not found. Then it checks for blanks following any continuation flag "\" and stops if one is found. Then it constructs a list of the targets specified in Makefile and stops if there are none or if there are too few targets to fulfill the request. If no targets are specified it lists the possible targets and stops. Next it checks each request given on the command line for the character "$" indicating a shell variable, and stops if there are any (a shell variable must be expanded into a list of individual targets for processing by mymake). If any of the targets to be made might be Fortran source and Makefile contains a default compilation rule, mymake checks the syntax of the rule. If mymake stops because any of these tests fails, it sets a return code of 1.

If all of these tests succeed mymake processes the requests one at a time in succession. If in examining the requests in succession mymake discovers one that does not match any target in Makefile, it reports that fact, lists the possible targets, and stops. If a request does match a target, /usr/bin/make is invoked to make the target, redirecting standard out to /tmp/output and standard error to /tmp/errors (this does away with the voluminous annoying noise output of /usr/bin/make). If /usr/bin/make fails, mymake reports that; if /tmp/output has 20 or fewer lines they are copied to the screen. If mymake stops for any of these reasons, it sets a return code of 2.

If /usr/bin/make succeeds, this script reports that fact and stops with a return code of 0.

DIAGNOSTICS

The return codes described above are summarized in this table.

0	all went well
1	a sanity check on the requests failed
2	/usr/bin/make failed to make a target

BUGS

When /usr/bin/make is invoked it determines which targets will be made and in which order. This permits parallel processing to update more than one target simultaneously, but it implicitly assumes that each target will be updated only by its own Makefile stanza. Once /usr/bin/make has determined that it will make a target, updating the target's modification time in some other stanza of the Makefile does not prevent it from being made. This behavior is not affected by the --jobs option. Thus it is possible for a target to get made even though it is more recent than its dependencies.

SEE ALSO

The man page for makemsg explains how to use it.

REPORTING ERRORS

To discard noise output mymake must also discard other outputs from programs that are run in Makefile, so useful error messages might be lost. Such messages can be written to the display by using makemsg.

If a program that is run in Makefile generates only an error message on standard-error, it can be printed by using makemsg like this. Here "filter" is a program that reads from standard-in, writes to standard-out, and issues error messages on standard-error.

```
filter < input > output 2>&1 | makemsg
```

If a program whose output is piped to makemsg stops without writing anything to the pipeline, the pipeline presents makemsg with an end-of-file on its standard-in so it stops with return code 0 and mymake is not interrupted. If the program writes a message to the pipeline, makemsg finds it on standard-in, writes the message, and stops with return code 1 to interrupt mymake. Thus the locution illustrated above passes on an error message and stops mymake if and only if "filter" writes something on standard-error.

If a program writing to a pipeline stops with a non-zero return code, the pipe completes anyway; it is the last program in the pipeline that determines its return code.

If a program that is run in Makefile generates only an error message on standard-out, it can be printed by using makemsg like this.

```
program | makemsg
```

If a program that runs in a Makefile produces output that should be presented to the user without interrupting mymake, it might be possible to capture the text in a file named fyle and then use makemsg "‘cat fyle‘" to display it. In this locution the command cat fyle is enclosed in backquotes within the double quotes.