Using MK (DOS) and MK32 (Win32)

Introduction

MK and MK32 (‘MK’) takes a file of dependencies (a 'makefile') and decides what commands have to be executed to bring the files up to date. These commands are either executed directly from MK or written to the standard output without executing them.

If no makefile is specified with a -f option, MK reads a file named `makefile', if it exists. If no target is specified on the command line, MK uses the first target defined in the first makefile.

OPTIONS

-f makefile

Use the description file `makefile'. A - as the makefile argument denotes the standard input.

-d Display the reasons why MK chooses to rebuild a target. All dependencies which are newer are displayed

-dd Display the dependency checks in more detail. Dependencies which are older are displayed, as well as newer.

-D Display the text of the makefiles as read in.

-DD Display the text of the makefiles and ‘default.mk’.

-e Let environment variables override macro definitions from makefiles. Normally, makefile macros override environment variables. Command line macro definitions always override both environment variables and makefile macros definitions.

-i Ignore error codes returned by commands. This is equivalent to the special target .IGNORE:.

-n No execution mode. Print commands, but do not execute them. Even lines beginning with an @ are printed. However, if a command line is an invocation of MK, that line is always executed.

-r Do not read in the default file ‘default.mk’.

-s Silent mode. Do not print command lines before executing them. This is equivalent to the special target .SILENT:.

-t Touch the target files, bringing them up to date, rather than performing the rules to reconstruct them.

macro=value

Macro definition. This definition remains fixed for the MK invocation. It overrides any regular definitions for the specified macro within the makefiles and from the environment. It is inherited by subordinate MK’s but act as an environment variable for these. That is, depending on the -e setting, it may be overridden by a makefile definition.

USAGE

Makefiles

The first makefile read is ‘default.mk’, which can be located anywhere along the PATH. It typically contains predefined macros and implicit rules.

The default name of the makefile is ‘makefile’ in the currentdirectory. An alternate makefile can be specified using one or more ’-f’ options on the

command line. Multiple '-f's act as the concatenation of all the makefiles in a left-to-right order.

The makefile(s) may contain a mixture of comment lines, macrodefinitions, include lines, and target lines. Lines may be continued across input lines by escaping the NEWLINE with a backslash (\).

Anything after a "#" is considered to be a comment, and is strippedfrom the line. Completely blank lines are ignored.

An include line is used to include the text of another makefile. It consists of the word "include" left justified, followed by spaces, and followed by the name of the file that is to be included at this line. Include files may be nested.

Macros

Macros have the form 'WORD = text and more text'. The WORD need not be uppercase, but this is an accepted standard. Later lines which contain $(WORD) or ${WORD} will have this replaced by 'text and more text'. If the macro name is a single character, the parentheses are optional. Note that the expansion is done recursively, so the body of a macro may contain other macro invocations.

e.g. FLINTSTONES = wilma and fred

RUBBLES = barney and betty

BEDROCK = $(FLINTSTONES) and $(RUBBLES)

'$(BEDROCK)' becomes 'wilma and fred and barney and betty'

Also note that whitespace around the equal sign is not relevant when defining a macro. The following four macro definitions are all equivalent:

MACRO = body

MACRO= body

MACRO =body

MACRO=body

Macros may be added to by using the '+=' notation. Thus

FLINTSTONES += pebbles and dino

would be (given the examples above) the same as

FLINTSTONES = wilma and fred and pebbles and dino

Special Macros

MAKE

This normally has the value "make". Any line which invokes MK temporarily overrides the -n option, just for the duration of the one line. This allows nested invocations of MK to be tested with the -n option.

MAKEFLAGS

This macro has the set of options provided to MK as its value. If this is set as an environment variable, the set of options is processed before any command line options. This macro may be explicitly passed to nested MAKEs, but it is also available to these invocations as an environment variable.

SUFFIXES

This contains the default list of suffixes supplied to the special target .SUFFIXES:. It is not sufficient to simply change this macro in order to change the .SUFFIXES: list. That target must be specified in your makefile.

There are several dynamically maintained macros that are useful as abbreviations within rules. It is best not to define them explicitly.

- $* The basename of the current target.
- $< The name of the current dependency file.
- $@ The name of the current target.

The $< and $* macros are normally used for implicit rules. They may be unreliable when used within explicit target command lines.

Targets

A target entry in the makefile has the following format:

target ... : [dependency ...]

[rule]

...

Any line which does not have leading whitespace (other than macro definitions) is a ‘target’ line. Target lines consist of one or more filenames (or macros which expand into same) called targets, followed by a semi-colon (:). The ':' is followed by a list of dependent files.

Special allowance is made on MSDOS for the colons which are needed to specify files on other drives, so for example, the following will work as intended:

c:foo.bar : a:fee.ber

If a target is named in more than one target line, the dependencies and rules are added to form the target's complete dependency list and rule list.

The dependents are ones from which a target is constructed. They in turn may be targets of other dependents. In general, for a particular target file, each of its dependent files is ‘made’, to make sure that each is up to date with respect to it's dependents.

The modification time of the target is compared to the modification times of each dependent file. If the target is older, one or more of the dependents have changed, so the target must be constructed. Of course, this checking is done recursively, so that all dependents of dependents of dependents of ... are up to date.

To reconstruct a target, MK expands macros, strips off initial whitespace, and either executes the rules directly, or passes each to a shell or COMMAND.COM for execution.

For target lines, macros are expanded on input. All other lines have macro expansion delayed until absolutely required.

Special Targets

.DEFAULT:

The rule for this target is used to process a target when there is no other entry for it, and no implicit rule for building it. MK ignores all dependencies for this target.

.DONE:

This target and its dependencies are processed after all other targets are built.

.IGNORE:

Non-zero error codes returned from commands are ignored. Encountering this in a makefile is the same as specifying -i on the command line.

.INIT:

This target and its dependencies are processed before any other targets are processed.

.SILENT:

Commands are not echoed before executing them. Encountering this in a makefile is the same as specifying -s on the command line.

.SUFFIXES:

The suffixes list for selecting implicit rules. Specifying this target with dependents adds these to the end of the suffixes list. Specifying it with no dependents clears the list. In order to add your own dependents to the head of the list, you could enter:

```
.SUFFIXES:
.SUFFIXES:      .abc $(SUFFIXES)
```

Rules

A line in a makefile that starts with a TAB or SPACE is a shell line or rule. This line is associated with the most recently preceding dependency line. A sequence of these may be associated with a single dependency line. When a target is out of date with respect to a dependent, the sequence of commands is executed. Shell lines may have any combination of the following characters to the left of the command:

@ will not echo the command line.

- MK will ignore the exit code of the command, i.e. the ERRORLEVEL of MSDOS. Without this, MK terminates when a nonzero exit code is returned.

+ MK will use COMMAND.COM to execute the command.

If the '+' is not attached to a shell line, but the command is a DOS command or if redirection is used (<, |, >), the shell line is passed to COMMAND.COM anyway.

Implicit Rules

Implicit rules are intimately tied to the .SUFFIXES: special target. Each entry in the .SUFFIXES defines an extension to a filename which may be used to build another file. The implicit rules then define how to actually build one file from another. These files are related, in that they must share a common basename, but have different extensions.

If a file that is being made does not have an explicit target line, an implicit rule is looked for. Each entry in the .SUFFIXES: list is combined with the extension

of the target, to get the name of an implicit target. If this target exists, it gives the rules used to transform a file with the dependent extension to the target file. Any dependents of the implicit target are ignored. In the following example, the .SUFFIXES: list is .c .y .l, and the target file is fred.o which does not have a target line. An implicit rule target '.c.o' is constructed and searched for. If it does not exist, the next suffix is tried. If the implicit rule target does exist, MK looks for a file 'fred.c'. If this file does not exist, the next extension is tried. If 'fred.c' does exist, then the associated rules are executed to create fred.o from fred.c, presumably invoking the C compiler.

If the next extension must be tried, MK reiterates the above with target '.y.o' and a file named 'fred.y', and potentially with '.l.o' and 'fred.l'.

EXAMPLES

This makefile says that pgm.exe depends on two files a.obj and b.obj, and that they in turn depend on their corresponding source files (a.c and b.c) along with the common file incl.h.

```
pgm.exe: a.obj b.obj
        $(CC) a.obj b.obj -o $@
a.obj:  incl.h a.c
        $(CC) -c a.c
b.obj:  incl.h b.c
        $(CC) -c b.c
```

The following makefile uses implicit rules to express the same dependencies.

```
pgm.exe: a.obj b.obj
        $(CC) a.o b.o -o $@
a.obj b.obj: incl.h
```

FILES

makefile	Current version(s) of make description file.
default.mk	Default file for user-defined targets, macros, and implicit rules.

DIAGNOSTICS

MK returns an exit status of 1 when it halts as a result of an error. Otherwise it returns an exit status of 0.

Badly formed macro

A macro definition has been encountered which has incorrect syntax. Most likely, the name is missing.

cannot open file

The makefile indicated in an include directive was not found or was not accessible.

Don't know how to make target

There is no makefile entry for target, none of MK's implicit rules apply, and there is no .DEFAULT: rule.

Improper Macro.

An error has occurred during macro expansion. The most likely error is a missing closing bracket.

rules must be after target

A makefile syntax error, where a line beginning with a SPACE or TAB has been encountered before a target line.

too many options

MK has run out of allocated space while processing command line options or a target list.