Make_TK Help Files

Dependencies.hlp

Dependency specifications have the form:

<target>: <dependents>

and can be followed by any number of command lines which begin with one or more spaces or tabs. Such commands are referred to as "explicit commands".

A dependency specification indicates that the commands should be executed if the target is found to be older than any of the dependents.

If more than one dependency line is specified for a target, then the dependency lists accumulate. Only one such dependency list may specify commands.

If a dependent occurs on a dependency line, then future references to that dependent implicitly refer to all of the other dependents on that earlier dependency line. This behavior may be overridden by specifying the target as a "double-colon" target, with the form:

<target> :: <dependents>

For succinctness, if the dependency list is short, commands may be specified on the target line by including a separating semicolon as:

<target> : <dependents> ; <commands>

This is especially useful for defining implicit commands.

```
*
* This is an example makefile meant to illustrate some of the
*  features and capabilities of make.   Comments are liberally
*  used to help explain what's going on.
*


*
* First, we define some macros.
*
*  ASM and LINK define command names for the implicit commands below.
*  AFLAGS and LFLAGS define command line switches.
*
ASM = rma     # Comments can be added at the ends of lines, too!
AFLAGS =
LINK = rlink
LFLAGS = -M=10   # This will give us extra memory in the compiled program.


*
* Now, we define some implicit commands.
*  By defining these, we simplify the rest of the makefile.
*
* Using so many macros is really overkill.  In normal use, we would
* simply put:
* .a.r: ; rma $< -o=$*.r
* .r:; rlink -M=10 $< -o=$*.r
*
*  $< is the name that make comes up with.  For .a.r, $< will just be
*    the old name with .a suffix instead of .r
*  $* is the prefix
*
.a.r: ; $(ASM) $(AFLAGS) $< -o=$*.r    # Make a .a file into a .r file
```

.r: ust li e with normal dependencies, we can put commands on the next line

I F A S < -o .r

e want to only loo at files with suffixes .a and .r

.S FFI S: Since there are no dependents, this will clear out the old list

.S FFI S: .a .r This means that implicit files will only have these exts.

The first target is the default one.

The at the end of the line means that the two lines should be
treated as one.
Since this target has more than one dependent, the implicit
rules won t generate the correct command line. So you must
specify a command line for this.

bigprogram: main.r subs .r subs .r subs .r subs .r subs .r subs .r subs .r
subs .r subs .r
rlin main.r subs .r subs .r subs .r subs .r subs .r
subs .r subs .r subs .r -o bigprogram

Since no commands are specified, the implicit rules will be invo ed
to generate a command line.

main.r: main.a

The obvious .a dependent will be used, since there is a .a.r rule

For this first one, for instance, will be e ual to "subs "
when the implicit command is generated, and < will be "subs .a".

```
subs  .r:
subs  .r:
subs  .r:
subs  .r:
subs  .r:
subs  .r:
subs  .r:
subs  .r:
```

This one needs special attention, so we specify commands for it.

```
subs  .r: oldstuff.c
    cc oldstuff.c -r
```

Macros.hlp

Macros are specified with the general form:

<macro> = <definition>

In commands or dependency lines, macros can be referenced by enclosing the macro name in parentheses or curly braces, and preceding it by a dollar sign. In the special case where the macro name is a single character, the parentheses may be omitted.

EXAMPLES

The following definition defines the macro OBJS to be a list of object files.

OBJS = file1.c file2.c file3.c

To specify that all object files depend on the common header file header.h, you could specify this as:

$(OBJS): header.h

SPECIAL MACROS

Several macros have special importance to the operation of make.

ODIR The ODIR macro determines the directory where make expects to find object files (i.e. files with no extension). By default, make looks in the current execution directory.

RDIR The RDIR macro determines where Make looks for files with a .r suffix. This is for compatibility with Tandy make. This support is partly hard-coded into Make, and partly

defined in the default rules.

The macro contains the root name i.e. without suffix of the current target. This is usually used with implicit rules see ules .

The macro contains the full name of the current target.

The macro contains a list of all files in the dependency list that were found to be newer than the target.

< The < macro contains the name of a dependent that was constructed by applying the implicit rules.

The following macros have importance by being defined as part of the internal implicit rules used by ma e.

The name of the compiler. y default it is "cc". sed in .c.r .a.r and .o.r rules.

F A S The command-line arguments to the compiler. Defaults to "-dOS ".

I The name of the in er. sed in the .r rule. Defaults to "cc"

F A S The command-line arguments to the lin er. Defaults to none.

Make.hlp

Syntax: Make <-opts> <target file> <macros>

  sage : Maintain, pdate and Regenerate roups of Programs

Opts : -b = don't use built in rules

-d = debug mode, print out the file dates in makefile

-f=<xxx> use <xxx> as makefile (default: makefile)

-i = ignore errors on commands, keep going

-n = don't execute commands, just print them out

-p = print out internal tables

- = check up-to-dateness of target

-s = silent mode, execute commands without echoing them

-t = update the dates without executing the commands

-u = do the make whether it needs it or not

- =<path> get list of files to make from stdin or path

---

a efile.hlp

The simplest makefile looks like the following:

```
program: program.c
    cc program.c
```

The first line specifies that the corresponding commands should be executed whenever "program" is older than "program.c". The second line is the commands to be executed to make "program" from "program.c".

See one of the topics below for more complete details and examples.

SpecialTargets.hlp

Several targets have special importance to Make. hen defined,
the dependents gain special attributes.

.S IXES The dependency list for this target specifies the suffixes
to be use in constructing implicit dependencies. This
list is tried in the order specified to attempt to construct
implicit dependencies. Specify .S IXES with no dependents
to clear the suffixes list. Repetitions accumulate.

.PRECIO S The dependency list for this target specifies files that
should not be deleted if an error occurs. ormally, if
an error occurs in constructing a specific target, that
target is deleted by make.

.SILE T If this target is specified, the make proceeds silently,
without echoing commands to the terminal. E uivalent to
the -s command line switch.
