$Id: asg1-perl-pmake.mm,v 1.76 2014-10-03 16:35:04-07 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps112-wm/Assignments/asg1-perl-pmake
URL: http://www2.ucsc.edu/courses/cmps112-wm/:/Assignments/asg1-perl-pmake/

1. Overview

Scripting is a style of programming whereby small programs are developed rapidly. This is also sometimes called rapid prototyping. Perl is a language which supports this particular programming paradigm very well because it is a very powerful and interpreted language. There is no need to do the usual compile-run cycle, since the program is compiled every time it is run.

One might claim that Perl is probably the best language for the job in almost any instance where the program can be written in, say, about an hour or less, and where the performance is not unacceptable. For large applications or those which require high performance, it is not suitable. Perl's most notable powerful features are powerful string manipulation facilities, pattern matching, and hash tables as primitive elements.

2. An implementation of make

In this assignment, you will use Perl to write a replacement for a subset of **`gmake`**.

NAME

pmake — perl implementation of gmake

SYNOPSIS

`pmake` [**`-d`**] [**`-n`**] [**`-f`** *makefile*] [*target*]

DESCRIPTION

The **`pmake`** utility executes a list of shell commands associated with each *target*, typically to create or update files of the same name. The *Makefile* contains entries that describe how to bring a target up to date with respect to those on which it depends, which are called prerequisites.

OPTIONS

The following options are supported. All options must precede all operands, and all options are scanned by **`Getopt::Std::getopts`** (perldoc).

`-d` Displays the reasons why make chooses to rebuild a target. This option prints debug information, or nothing at all. Output is readable only to the implementor.

`-n` Non-execution mode. Prints commands, but does not execute them.

`-f`*Makefile*

Specifies the name of the Makefile to use. If not specified, tries to use **`./Makefile`**. If neither of those files exists, exits with an error message.

OPERANDS

The following operand is recognized.

target

An attempt is made to build each target in sequence in the order they are given on the command line. If no target is specified, the first target in the *makefile* is built. This is usually, but not necessarily, the target **`all`**.

EXIT STATUS

0 No errors were detected.

1 An error in the makefile was detected, one that did not come from the execution of a subprocess.

>0 An error was detected which caused **pmake** to stop execution, because of an error return from the subprocess. In this case, the subprocess's exit code is returned.

MAKEFILE SYNTAX

Generally, whitespace delimits words, but in addition, punctuation is recognized as well. Each line of input is a comment, an empty line, a dependency, or a command.

Any line with begins with a hash, possibly preceded by whitespace (spaces and tabs) is ignored. Empty lines consisting only of whitespace are also ignored.

macro **=** *value*
Macro definitions are kept in a symbol (hash) table, to be substituted later.

target . . . : *prerequisite* . . .
Each target's time stamp is checked against the time stamps of each of the prerequisites. If the target or prerequisite contains a percent sign (**%**), it is substituted consistently. If any target is obsolete, the following commands are executed. A target is obsolete if it is a file that is older than the prerequisites or does not exist. A prerequisite is either a file or another target. If a file, its time stamp is checked. If not, the target to which is refers is made recursively. No target is made more than once.

command
A command is any line for which the first character is a tab . The line is echoed to **STDOUT** before executing the command. The line is then passed to the **system** function call for execution by the shell. The resulting exit code is then tested. If it is non-zero, **pmake** exits at that point with that exit code. If the tab is followed by a minus sign (**-**) and white space, the exit code is treated as 0. The minus sign is not passed to the shell.

MACROS

Whenever a dollar sign appears in the input file, it represents a macro substitution. Macros are substituted from innermost to outermost braces. If a dollar sign is followed by any character except a left parenthesis (**(**) or left brace (**{**) that one character is the macro name. Otherwise, the contents of the parentheses or braces are replaced.

$$ Represents the dollar sign itself.

$< Represents the first file specified as a prerequisite.

$@ Represents the first file specified as a target.

`${...}` The contents of the braces are substituted with the value of the macro name, which may be multiple characters, not including a closing brace.

```
 1  // $Id: sigtoperl.c,v 1.4 2011-12-20 17:08:31-08 - - $
 2
 3  #define _GNU_SOURCE
 4
 5  #include <signal.h>
 6  #include <stdio.h>
 7  #include <stdlib.h>
 8  #include <string.h>
 9
10  int main (void) {
11     int sig;
12     system ("echo '#' `uname -srp`");
13     printf ("my %%strsignal = (\n");
14     for (sig = 0; sig < _NSIG; ++sig) {
15        char *strsig = strsignal (sig);
16        if (strsig == NULL) continue;
17        printf ("%5d => \"%s\",\n", sig, strsig);
18     }
19     printf (");\n");
20     return EXIT_SUCCESS;
21  }
22
```

Figure 1. `sigtoperl.c`

3. Commentary

Here are some hints that will be useful in familiarizing yourself with Perl and how to perform certain kinds of coding.

(a) There are a lot of Perl scripts that may be used as examples in the directory **`/afs/cats.ucsc.edu/courses/cmps112-wm/bin`**, which is a symlink to **`/afs/cats.ucsc.edu/courses/cmps012b-wm/bin`**.

(b) The function **`system`** will pass a comment string to the shell and set the variable **`$?`** to the **`wait`**(2) return value. If the termination signal is 0 (bits 6...0), then the program exited normally and bits 15...8 contain the **`exit`**(2) status returned by the program. Otherwise, bits 6...0 contain the signal that caused the program to terminate, and bit 7 indicates whether or not core was dumped. The following code can be used to extract this information:

```
my $term_signal = $? & 0x7F;
my $core_dumped = $? & 0x80;
my $exit_status = ($? >> 8) & 0xFF;
```

```
 1  # Linux 2.6.32-431.29.2.el6.x86_64 x86_64
 2  my %strsignal = (
 3       0 => "Unknown signal 0",
 4       1 => "Hangup",
 5       2 => "Interrupt",
 6       3 => "Quit",
 7       4 => "Illegal instruction",
 8       5 => "Trace/breakpoint trap",
 9       6 => "Aborted",
10       7 => "Bus error",
11       8 => "Floating point exception",
12       9 => "Killed",
13      10 => "User defined signal 1",
14      11 => "Segmentation fault",
15      12 => "User defined signal 2",
16      13 => "Broken pipe",
17      14 => "Alarm clock",
18      15 => "Terminated",
19      16 => "Stack fault",
20      17 => "Child exited",
21      18 => "Continued",
22      19 => "Stopped (signal)",
23      20 => "Stopped",
24      21 => "Stopped (tty input)",
25      22 => "Stopped (tty output)",
26      23 => "Urgent I/O condition",
27      24 => "CPU time limit exceeded",
28      25 => "File size limit exceeded",
29      26 => "Virtual timer expired",
30      27 => "Profiling timer expired",
31      28 => "Window changed",
32      29 => "I/O possible",
33      30 => "Power failure",
34      31 => "Bad system call",
35      32 => "Unknown signal 32",
36      33 => "Unknown signal 33",
68  );
```

Figure 2. Partial output from **`sigtoperl`**

(c) Figure 1 shows a C program (**`code/sigtoperl.c`**) which prints out a description of all of the signals, some of the output of which on this machine is shown in Figure 2. The complete output of this program may be inserted into your Perl program.

(d) Do ***not*** cut and paste from the text file or the pdf. If you use **`vi`**, you can insert it into your program with the command

```
.!./sigtoperl
```

if the binary executable is in the current directory. Presumably other editors have similar facilities.

(e) Use the function **system** to run the command. **$?** is the **wait**(2) exit status.

(f) Keep all macros in a hash table.

(g) To extract the innermost macro substitution, the following pattern will avoid nested macros: **\${[^}]+}**. Alternately, you may wish to parse macro lines into an AST matching braces. Remember that regular expressions don't handle matched structures but context free grammars do.

(h) Keep each target in a hash with the prerequisites and commands as a reference to a list. Hashes are used in Perl to represent structs. Thus, the following will point **$p** at a struct with two fields:

```
$p = {FOO=> 3, BAR=> [1, 2, 3]}
```

(i) The **stat** function returns a list of file attributes. The modification time is the value of interest when comparing time stamps on files. See **perlfunc**(1).

```
@filestat = stat $filename;
my $mtime = $filestat[9];
```

(j) Look at the subdirectories **.score/test*** and see what **gmake** does with them.

4. What to submit

Submit one file, specifically called **pmake**, which has been **chmod**ed to executable (**+x**). The first line must be a hashbang for Perl. Also, use **strict** and **warnings**.

```
#!/usr/bin/perl
# Your name and username@ucsc.edu
use strict;
use warnings;
```

Your name and RCS **Id** string must come ***after*** the hashbang line. Grading will be done by naming it as a shell script. Do not run it by typing the word **perl** as the first word on the command line.

If you are doing pair programming, submit **PARTNER** as required by the pair programming instructions in **cmps112-wm/Syllabus/pair-programming**.