

\$Id: asg4-perl-pmake.mm,v 1.32 2019-02-07 14:22:07-08 - - \$

PWD: /afs/cats.ucsc.edu/courses/cmcs112-wm/Assignments/asg4-perl-pmake

URL: <http://www2.ucsc.edu/courses/cmcs112-wm/:/Assignments/asg4-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.

The `make(1)` utility determines automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them. This project will also enhance your knowledge of `make` and `Makefiles`, as presented in prerequisite courses. Every programmer should have a detailed knowledge of `make`.

## 2. An implementation of a subset of `gmake`

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

### NAME

`pmake` — perl implementation of a subset of `gmake`

### SYNOPSIS

`pmake` [-d] [*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` (perl doc).

**-d** Displays debugging information. Output is readable only to the implementor.

### 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`.

### FILES

Reads the file called *Makefile* in the current directory. If the file does not exist, `pmake` exits with an error message.

### EXIT STATUS

0 No errors were detected.

- 1 An error in the **Makefile** was detected. Or if any subprocess returned a non-zero exit status or failed on a signal, and the command was not preceded by the minus sign (-) marker.

## 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 which 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 it 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 status and signal is then tested. If either is non-zero, **pmake** exits at that point.

@ *command*

Behaves like *command*, except that the command is not echoed to **STDOUT** before being executed.

- *command*

Behaves like *command*, except that a non-zero exit status or signal does not cause **pmake** to exit at that point.

## 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 brace that one character is the macro name. Otherwise, the characters between the braces constitute the name of the macro.

- \$\$ 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.

### 3. Commentary

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

- (a) The directory `/afs/cats.ucsc.edu/courses/cmcs112-wm/bin` contains examples of Perl scripts. And the subdirectory `code/` of this directory contains relevant code.
- (b) The function `system` will pass a command 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;
```

- (c) A C++ program `code/sigtoperl.cpp` prints out a description of all of the signals. The output is in `code/perlsignals.out`. This output may be inserted into your Perl program.
- (d) Use the function `system` to run the command. `$?` is the `wait(2)` exit status. The notation `wait(2)` refers to the manual page in section 2 of the manual. The command may be read with the command

```
man -s 2 wait
```

- (e) Keep all macros in a hash table.
- (f) 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.
- (g) 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]}
```

- (h) 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];
```

- (i) 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**. Your name 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. The first few lines are :

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

If you are doing pair programming, submit **PARTNER** as required by the pair programming instructions in `cmcs112-wm/Syllabus/pair-programming`.