Syntax

nsh is somewhat similar to existing shells such as csh, bash, ksh, and tcsh. However, it lacks many of the features of those shells. It also has a different command syntax and a different command set.

nsh takes commands from standard input and directs all its output to standard output.

Like other shells, nsh accepts both built-in commands and program-control commands.

Variable definition and substitution

When nsh encounters a variable definition, such as setvar variable "value", it should store the name and definition. Later, when it encounters $variable either as a token or within a string, nsh should replace $variable with its stored definition. If there is no stored definition, nsh should output an error message but use the empty string as its substitution. It is valid to modify the definition of a variable by a later definition.

Built-in Commands

nsh does not need to verify that built-in commands have the right number of parameters, but all required parameters must be present.

- `set` *`variable value`*
  Here, *variable* is any reasonable variable name (starting with a letter, continuing with letters or numbers), and *value* is a token. It is conventional for users to represent variables in ALL CAPITALS, but nsh does not enforce this convention. Variable names are case-sensitive, that is, home and HOME represent different variables. The effect of this command is to associate the variable name with the given value in a data structure inside nsh.
- `prompt` *`newPrompt`*
  Set the shell prompt to *newPrompt*, which is a token. Do not add or subtract spaces at the end of the new prompt. The initial prompt in nsh is nsh > .
- `dir` *`directoryName`*
  This command changes the current directory to *directoryName*. The required parameter *directoryName* may be either absolute (starting with /) or relative (not starting with /).
- `procs`
  nsh lists all processes running in the background.
- `done`
  nsh exits.

Program-control commands

- do *cmd* [*param* ...](The brackets indicate "optional" and the dots indicate "and more if desired".) The user submits a do command to execute a program. The keyword do must appear as the first token on the command line, followed by the command to be executed: *cmd* is a token that specifies the filename of the program the user wants to execute. It is followed by zero or more tokens specifying parameters. nsh should wait for the program to finish before it prompts for and accepts the next command.
  If cmd starts with a / character, it is a full path name starting at the root of the filesystem.
  A cmd that starts with ./ is a path name starting in the current directory.
  Otherwise, nsh looks for cmd in a list of directories indicated by the variable PATH, whose value is a colon-separated list of directories. By default, PATH has value /bin:/usr/bin; the user can change that value by assigning a new value to the variable PATH.

- back *cmd* [*param* ...]The back command is identical to the run command, except that the process running the command should be in the background.

- tovar *variable cmd* [*param* ...]The tovar command executes the program cmd along with its parameters, if any, just like the do command. However, nsh absorbs the standard output of the program and assigns it as the value of variable specified by the second token.

Invalid commands

If the user inputs an invalid command, nsh should simply print an error message (to stderr) and prompt for the next command. If nsh encounters a problem that it cannot surmount, it should exit with status -1.

Sample commands

```
     set PATH "/bin:/usr/bin" # this is the default setting
     set HOME "/home/mylogin" # a useful shorthand
     set Myfile "/etc/hosts" # the quotes are unnecessary but
valid
     set Filebase "paper"
     set ShowTokens 1 # a special token, described later
     do echo "my path is $PATH" # should output "my path is /
bin:/usr/bin"
     set /usr/bin # no output expected
     do ls awk # prints "awk".
     set ../misc/oldstuff # prints an error message.
     set $HOME
     do /usr/bin/wc -l $Myfile # prints "9 /etc/hosts".
     back wc -l $Myfile $Myfile # prints three lines
     back /bin/sleep 10 # immediately gives a new prompt
     procs # should list at least the /bin/sleep, maybe the wc
process
     tovar TextFile /bin/echo -e $Filebase "\b.txt"
```

```
do echo $TextFile # prints "paper.txt"
do /bin/ls
do ls -l -F -g -s /tmp
done
```