NAME

sish — a simple shell

SYNOPSIS

sish [**-x**] [**-c** *command*]

DESCRIPTION

sish implements a very simple command-line interpreter or shell. It is suitable to be used interactively or as a login shell. It only implements a very small subset of what would usually be expected of a Unix shell, and does explicitly not lend itself as a scripting language.

OPTIONS

The following options are supported by **sish**:

-c *command*
: Execute the given command.

-x
: Enable tracing: Write each command to standard error, preceeded by '+'.

DETAILS

sish allows for the execution of the given commands with a minimal amount of niceties.

When **sish** is invoked and no commands are passed via the **-c** flag, it will print a simple prompt to stdout. It then reads commands from the controlling terminal, executing them as one would expect.

Lexical Structure

The shell reads input in terms of lines and breaks it up into words at whitespace (blanks and tabs), and at certain sequences of characters that are special to the shell called ‘‘operators’’. There are two types of operators: control operators and redirection operators:

Control operators:

```
& | <newline>
```

Redirection operators:

```
< > >>
```

Redirection

sish supports the following three input-/output- redirection operators:

> file
: Redirect standard output to file.

>> file
: Append standard output to file.

< file
: Redirect standard input from file.

Note: multiple redirection operators can be given in the same command. If multiple operators of the same type are specified, then the last one will ultimately take effect.

Pipeines

A pipeline is a sequence of one or more commands separated by the control operator '|'. The standard output of all but the last command is connected to the standard input of the next command. The standard output of the last command is, absent any redirection operators, the controlling terminal.

Background Commands -- &

If a command is terminated by the control operator ampersand (&), the shell executes the command asynchronously -- that is, the shell does not wait for the command to finish before prompting the user for the next command.

Builtins

sish supports the following builtins (which are taking precedence over any non-builtin commands):

cd [*dir*] Change the current working directory. If *dir* is not specified, change to the directory specified in the HOME environment variable. If that is unset, change to the user's home directory as determined via getpwuid(3).

echo [*word ...*] Print the given word(s), followed by a '\n'. The following special values are supported:

$? The exit status of the last command.

$$ The current process ID.

exit Exit the current shell.

Command Execution

If a sequence of words does not begin with a builtin, **sish** will attempt to execute it as a command, possibly utilizing the current PATH.

Environment

sish sets the following environment variables:

SHELL the path of the executable of **sish**

As noted above, **sish** uses the HOME environment variable when the shell builtin *cd* is invoked without any arguments.

EXAMPLES

The following sequence of commands shows common usage of **sish**:

```
$ sish
sish$ echo $$
6465
sish$ ls
file1   file2
sish$ ls | wc -l
    2
sish$ echo $?
0
sish$ echo Writing a shell is fun! > /tmp/out
sish$ < /tmp/out cat
Writing a shell is fun!
sish$ find / >/dev/null &
sish$
sish$ aed -e <file >file.enc
sish$ cmd | sort | uniq -c | sort -n
 121 foo
 304 bar
sish$ something
something: command not found
sish$ echo $?
127
sish$ rm /etc/passwd
rm: /etc/passwd: Permission denied
sish$ echo $?
```

```
1
sish$ exit
$ sish -c date
Sat Nov 29 21:18:07 EST 2014
$ sish -c "date +%s"
1576614990
$ sish -x
sish$ ls
+ ls
file1    file2
sish$ ls | wc -l
+ ls
+ wc -l
    2
sish$ cd /tmp
+ cd /tmp
sish$ pwd
+ pwd
/tmp
sish$ cd
+ cd
sish$ pwd
+ pwd
/home/jschauma
sish$ exit
+ exit
$
```

EXIT STATUS

sish returns the exit status of the last command it executed or a status of 127 if the given command could not be executed for any reason.

SEE ALSO

bash(1), ksh(1), sh(1), execve(2), fork(2)

HISTORY

Writing a simple shell has been a frequent assignment in many Unix programming classes. This particular version was first assigned in the class *Advanced Programming in the UNIX Environment* at Stevens Institute of Technology by Jan Schaumann ⟨jschauma@stevens.edu⟩ in the Fall of 2014.