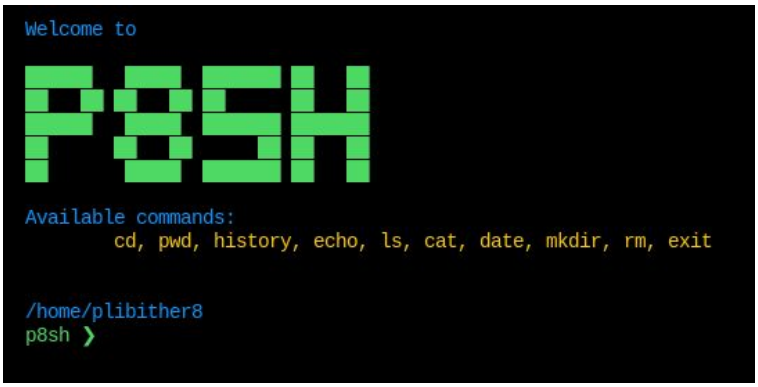
CSE231: Operating Systems

Assignment 1, Part 2: Basic Shell

Name: Mihir Chaturvedi
Roll no.: 2019061
Section: A
Branch: CSE

Introduction



I created this shell, namely **p8sh**, with intent of implementing a simple, but not too simplistic *nix based shell. As part of this assignment, the following ten commands have been implemented: (internal) `cd`, `echo`, `history`, `pwd`, `exit`; (external) `cat`, `date`, `ls`, `mkdir` and `rm`.

Features

Along with the command specific features, these are some overall distinctive shell features:

- Colorful and descriptive output: the prompt's color indicates the previous commands exit status (and thus, error status): green indicates a successful run, red indicates that the command exited with an error.
- Adherence to all sorts of input formats for each command (not options). All commands that support multiple input arguments are supported.

- Short options can be compactly written together, or in long-form. All permutations/repetitions are respected. Example, any of the following can be used:
  `--all --verbose <=> -a -v <=> -av <=> -va`
- Supports wildcard expansion using the asterisk. Example: “cat *.c’ will match all ‘.c’ extension files in and concat them.
- Supports environment variable expansion in the arguments, home directory alias (the tilde: ~), quoted arguments, backslash escapes, etc.

Overall Assumptions and Caveats

- Disabling option parsing using ‘--’ is not supported. All tokens leading with a hyphen will be considered options will be parsed as options, appropriately.
- Command length will not exceed 300 characters.
- There are more “magic numbers” like these for string length, for which it was becoming impractical to account for variable length.

Organisation

The directory structure of the source is as below:

```
.
├── external
│   ├── cat.c
│   ├── date.c
│   ├── ls.c
│   ├── mkdir.c
│   └── rm.c
├── internal
│   ├── cd.h
│   ├── echo.h
│   ├── history.h
│   ├── pwd.h
│   └── exit.h
├── Makefile
├── global.h
└── p8sh.c
```

- Source code (header files) for all internal commands are placed in the `/internal` directory.
- Source code (header files) for all external commands are placed in the `/external` directory.

- p8sh.c is the main, parent file of the shell containing the `main()` function. It includes the internal commands' header files, along with other libraries. Along with this, forking the process and executing the external commands' executables also takes place here.
- global.h holds the ANSI escape codes for colorful/formatted terminal output, along with global variable definitions.
- The Makefile generates binaries for external commands that are placed inside a '/bin' directory. One main executable is generated for p8sh.c.

Code Style

Wherever possible, I have attempted to adhere to CS50's style guide for C programming: Style Guide for C — CS50 Docs. Exceptions have been made at places where I thought appropriate.

Commands

`cd`

Having made my own parses for path resolution, I have been able to support many variations including paths that are a mix of absolute and relative tokens, normalisation of multiple slashes, etc. So in essence, all these are these are allowed:

- `cd ///home///`
- `cd /..`
- `cd /home/user/../`
- `cd ~//Downloads/./`

Options

- -P use the physical directory structure without following
- -L force symbolic links to be followed: resolve symbolic links in DIR after processing instances of `..' (default)

Errors handled

- "no such file or directory", when a user attempts to `cd` into a nonexistent entry.
- "not a directory", when a user attempts to `cd` into anything that isn't a directory.
- "invalid option", when a user provides an invalid option.
- " too many arguments", when a user enters more than one entry.

`echo`

Prints the multiple arguments provided to it, space separated. Respects quoted strings. Only valid options are entered, invalid options are echoed. Options must be at the start of the command only.

Options

- -n do not output the trailing newline
- -E disable interpretation of backslash escapes (default)

Corner cases

- Don't parse options after the first non-option token has been encountered.
- Don't handle invalid options.

Errors handled

`echo` itself does not have, or require to throw, errors. Any input is handled natively.

`pwd`

Prints the logical path of the current working directory, or physical unless specified.

Options

- -P avoid all symlinks
- -L resolve symbolic links (default)

Errors handled

- "invalid option", when a user provides an invalid option.
- "too many arguments" is NOT handled. This is in accordance with how the Bash shell handles this.

`history`

Prints the entire contents of the history file (line-numbered), ie, the saved history of command over different sessions. If a numerical non-option argument 'n' is specified, it prints the last 'n' history entries.

Options

- -c clear the history list by deleting all of the entries
- -d offset delete the history entry at position OFFSET

Errors handled

- “too many arguments”, when a user provides any argument after the numerical N argument.
- “-d: option requires an argument”, when a user does not provide an argument after the ‘-d’ option.
- “numeric argument required”, when a user doesn’t provide a numeric argument N.
- “history position out of range”, when the supplied number to ‘-d’ argument does not correspond to any position in the history file.
- Any other miscellaneous file-handling error while `fopen()` -ing the history file.

`exit`

Exits shell with 0 exit code, unless explicitly supplied one as the first argument.

Options

<No options>

Errors handled

- “numeric argument required”, when a user doesn’t provide a numeric argument N.
- “too many arguments”, when a user provides any argument after the numerical exit_code argument.

`ls`

List files and directories present in either the working directory, if no argument supplied, or those in the list of arguments supplied.

Options

- -a, --all do not ignore entries starting with .
- -i, --inode print the index number of each file
- -1 list one file per line

Errors handled

- “unrecognized option”, when a user provides an invalid option’s longform.

- “invalid option”, when a user provides an invalid option.
- “no such file or directory”, when a user attempts to `ls` a nonexistent entry.

Corner cases

- If multiple entries are specified, print the entry’s name first, and then its listing in the next line.
- If `all` is specified, first print the parent and current directory identifiers: ‘..’ and ‘.’, and then print the rest of the files in alphabetical order.
- If a regular file is specified, just print its name.

`cat`

Concatenate files and print them to the standard output. When no file is specified, or when file is ‘-’, read from standard input.

Options

- -E, --show-ends display $ at end of each line
- -n, --number number all output lines

Errors handled

- “unrecognized option”, when a user provides an invalid option’s longform.
- “invalid option”, when a user provides an invalid option.
- “is a directory”, when a user provides ‘cat’ with a directory name,
- “no such file or directory”, when a user attempts to `cat` a nonexistent entry.

Corner cases

- Handling ‘-’ input, or generally reading from stdin proved to be challenging since I wanted to exit the input-loop when the user forces an EOF through Ctrl+D.

Assumptions

- The user will not attempt pass ‘-’ as a file twice in one operation. Rationale: the EOF is passed into the subsequent stdin, and no input is taken.
- User does not pass a binary file.

`mkdir`

Create directories from those supplied in the arguments.

Options

- -m, --mode=MODE set file mode (as in chmod)
- -v, --verbose print a message for each created directory

Errors handled

- “unrecognized option”, when a user provides an invalid option’s longform.
- “invalid option”, when a user provides an invalid option.
- “missing operand”, when a user does not provide any pathname argument.
- “invalid mode”, when an invalid mode is passed through -m/--mode option.
- “cannot create: Permission denied", when the user/program does not have sufficient permissions to create a new directory in the working directory.
- “cannot create: File exits", when a file or directory of the same name already exists.
- “cannot create: No such file or directory", when a component of the path prefix specified by path does not name an existing directory or path is an empty string.

`rm`

Remove/delete files that are supplied in the arguments.

Options

- -f, --force ignore nonexistent files and arguments, never prompt
- -v, --verbose explain what is being done

Errors handled

- “unrecognized option”, when a user provides an invalid option’s longform.
- “invalid option”, when a user provides an invalid option.
- “missing operand”, when a user does not provide any pathname argument.
- “cannot create: Permission denied", when the user/program does not have sufficient permissions to create a new directory in the working directory.
- “cannot create: Is a directory", when the pathname is a directory.
- “cannot create: No such file or directory", when a user attempts to `rm` a nonexistent file.

`date`

Print the system date, in the specified format (or not).

Options

- -u, --utc, --universal print Coordinated Universal Time (UTC)
- -R, --rfc-email output date and time in RFC 5322 format. Example: Mon, 14 Aug 2006 02:34:56 -0600

Errors handled

- “unrecognized option”, when a user provides an invalid option’s longform.
- “invalid option”, when a user provides an invalid option.
- “extra operand”, when a user provides any other argument other than the format string.
- “invalid date”, when the user does not provide a valid format specifier string (it should ideally start with a “+”).
- “multiple output formats specified”, when multiple output formats are specified (the format string coupled with the -R option).

Caveats

- When the “-u” option is passed, the timezone reads as GMT instead of UTC.

Test cases

For the test cases, I have not included incorrect, invalid commands that might return a non-zero exit code. This will increase the verbosity of the write-up. The error handling can be manually checked or demonstrated when required.

The test mostly takes place in the `test` directory. I have assumed a very simple structure, with directories being ‘test/bin’, ‘test/hello’, ‘test/world’, the latter two containing some empty files. There are two multiline files ‘file1.txt’ and ‘file2.txt’ for demonstration.

```
/home/plibither8/test
p8sh ❯  cd ~
```

```
/home/plibither8
p8sh ❯  cd /
```

```
/
p8sh ❯  cd ./home/test
p8sh: cd: no such file or directory: ./home/test

/
p8sh ❯  cd ./home/plibither8/test

/home/plibither8/test
p8sh ❯  ls -a1i
10230647 .
9837020 ..
10247070 .p8sh_history
10247040 bin
10247053 file1.txt
10247432 file2.txt
10230649 hello
10247431 p8sh
10247007 world

/home/plibither8/test
p8sh ❯  cd hello/../hello/./.

/home/plibither8/test/hello
p8sh ❯  ls
night    of    out    the    world
/home/plibither8/test/hello
p8sh ❯  rm --verbose out
Removed 'out'

/home/plibither8/test/hello
p8sh ❯  ls
night    of    the    world
/home/plibither8/test/hello
p8sh ❯  cd ..

/home/plibither8/test
p8sh ❯  history
     1
     2
     3  cd ~
     4  cd /
     5  cd ./home/test
     6  cd ./home/plibither8/test
```

```
     7  ls -a1i
     8  cd hello/../hello/./.
     9  ls
    10  rm --verbose out
    11  ls
    12  cd ..
    13  history

/home/plibither8/test
p8sh ❯  history 5
    10  rm --verbose out
    11  ls
    12  cd ..
    13  history
    14  history 5

/home/plibither8/test
p8sh ❯  history -d 12

/home/plibither8/test
p8sh ❯  history 5
    11  ls
    12  history
    13  history 5
    14  history -d 12
    15  history 5

/home/plibither8/test
p8sh ❯  mkdir -v --mode=0775 mihir
mkdir: created directory 'mihir'

/home/plibither8/test
p8sh ❯  ls *
bin:
cat    date    ls    mkdir    rm

file1.txt

file2.txt

hello:
night    of    the    world
```

```
mihir:


p8sh

world:
covers    me    that

/home/plibither8/test
p8sh ❯  cat -nE file1.txt file2.txt
     1  __file1 here!_$
     2  hello$
     3  world$
     4  my name is$
     5  mihir$
     6  __this is file 2__$
     7  this$
     8  is a$
     9  test$
    10  file$

/home/plibither8/test
p8sh ❯  date -R
Wed, 30 Sep 2020 18:30:08 +0530

/home/plibither8/test
p8sh ❯  date -u "+%H:%M:%S"
13:00:22

/home/plibither8/test
p8sh ❯  date -u "%S"
date: invalid date '%S'

/home/plibither8/test
p8sh ❯  echo "hello world"        nice
hello world nice

/home/plibither8/test
p8sh ❯  echo -n "nice"
nice
/home/plibither8/test
p8sh ❯  pwd -P
/home/plibither8/test
```