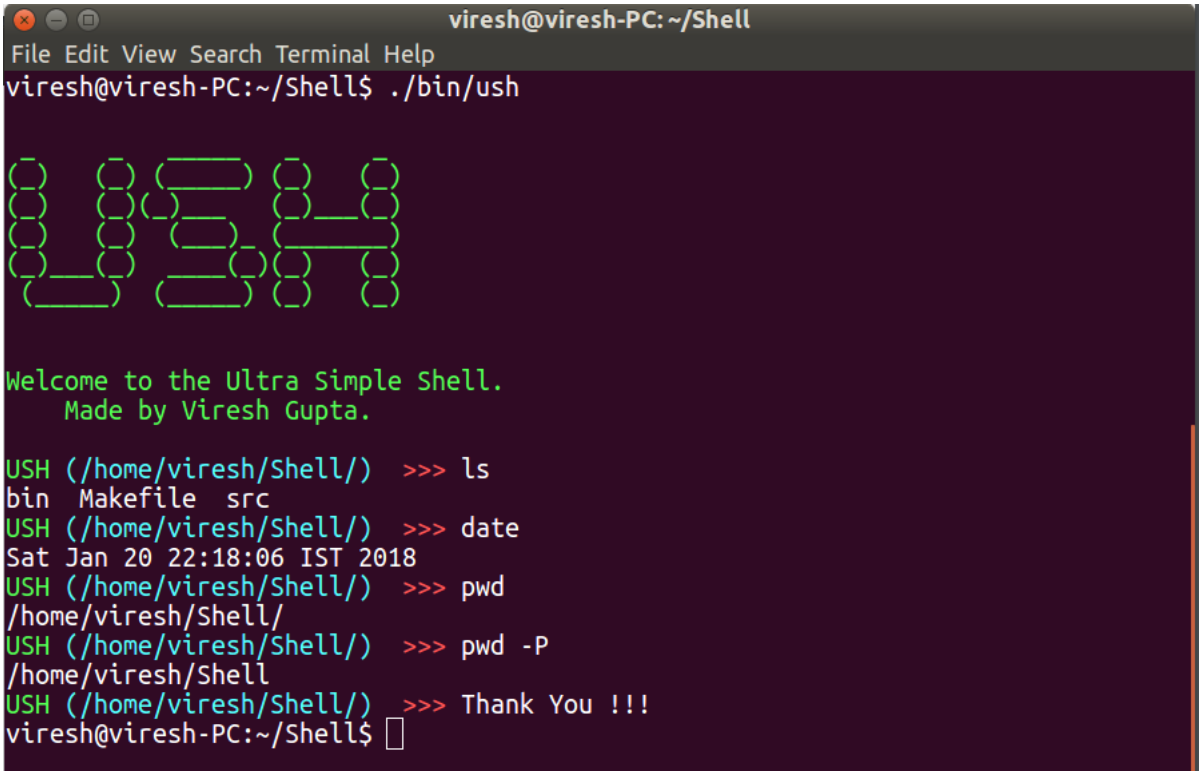
Assignment 0 - USH

Viresh Gupta | 2016118

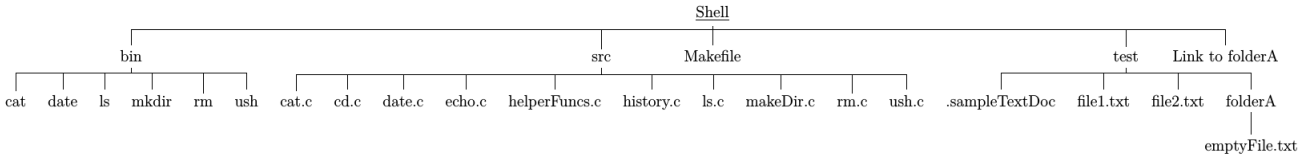
January 23, 2018

Contents

Introduction



Structure of the testing environment



Assumptions made:
The shell assumes that it will be used only on a linux based system that allows current binary path resolution via /proc/self/exe
Further the shell will assume that the shell binary and all the related binaries for external commands are placed all together. (No concept of path variable).

1 Compiling the shell

Commands

From the directory 'Shell', ensure that a bin folder exists (may or may not be empty), then execute the makefile as follows:

```
viresh@viresh-PC:~/Shell$ make
gcc src/ush.c -o bin/ush
gcc src/ls.c -o bin/ls
gcc src/date.c -o bin/date
gcc src/cat.c -o bin/cat
gcc src/makeDir.c -o bin/mkdir
```

```
gcc src/rm.c -o bin/rm
```

The makefile will place all binaries inside the bin folder.
Thus all source files are in src folder and all binaries inside bin folder. Now the shell can be executed as:

```
viresh@viresh-PC:~/Shell$ ./bin/ush


 _     _  _____  _     _
(_)   (_)(_____)(_)   (_)
(_)   (_)(_)___   (_)___(_)
(_)   (_) (___)_ (_______)
(_)___(_)  ____(_)(_)   (_)
 (_____)  (_____)(_)   (_)


Welcome to the Ultra Simple Shell.
    Made by Viresh Gupta.

USH (/home/viresh/Shell/)  >>>
```

Some noteworthy points:
The shell will set it's current path to the location from where it is invoked, but the default directory (∼) will always point the location of the binary.

There is partial support for names with spaces, and no support for names starting with a - (hypen).

Five internal and Five external commands have been implemented.
For all external commands, it is assumed that their binary will be at the same location as the shell's binary.

Also the shell is unaffected by empty lines.

Also note that the environment used here is a little bit different that the one generated via make file. Hence the commands shown here might need some adjustments, Thus it is suggested to use the commands in the cases.txt document provided.

Also in case you do not wish to generate the test environment, just use "make only_executables" to only make executables

2 Internal Commands

USH implements the following five internal commands:

-> cd — Options : -L and -P

-> echo — Options : -n and -e

-> history — Options : -c and -d

-> pwd — Options : -L and -P

-> exit Options : -s

The code for these internal commands lies within the USH binary itself, hence even if for any reason the other binaries are lost, these commands are guaranteed to work.

2.1 cd

The cd command changes the current working directory of the shell.
Format : $cd\ [-PL]\ [directory\ path]$
Options available:
-P : Resolve all symlinks and cd into an absolute path.
-L : (default) Do not resolve any symlinks.
Important Shortcuts:
cd .. - Lands you into the parent directory. You can nest ..'s. For e.g cd ../../ will take two levels up.
cd $\sim$ - This will cd into the default directory, which is the one where the shell binary resides.
Commands to test cd:

```
USH (/home/viresh/Shell/)  >>> cd Link to folderA
USH (/home/viresh/Shell/Link to folderA/)  >>> cd -P .
USH (/home/viresh/Shell/test/folderA)  >>> cd ..
USH (/home/viresh/Shell/test/)  >>> cd -L ../Link to folderA
USH (/home/viresh/Shell/Link to folderA/)  >>> cd ..
USH (/home/viresh/Shell/)  >>>
```

Note: Error generated on invalid arguments.

2.2 echo

The echo command outputs the arguments as it is.
Format : $echo\ [-ne]\ <args>$
Options available:
-n : Do not print the trailing newline character.
-L : Interpret and properly render the escape sequences.

The following escape sequences are supported:
$\backslash n, \backslash b, \backslash v, \backslash t, \backslash r, \backslash 0, \backslash\backslash, \backslash a$.

Commands to test echo:

```
USH (/home/viresh/Shell/)  >>> echo -n LOL\n
LOL\nUSH (/home/viresh/Shell/)  >>> echo -ne LOL\n
LOL
USH (/home/viresh/Shell/)  >>>
```

Note: Additional / invalid arguments are ignored. Also you cannot echo a string that starts with a hyphen (-).

2.3 history

The history command outputs the history of the shell session.
(This history is saved in .ush_history after session exit)
Format : $history\ [-cd]\ <args>$
Options available:
-c : Clear complete history.
-d <offset> : Remove the history command at the given offset value. Error if invalid offset.

Commands to test history:

```
USH (/home/viresh/Shell/)  >>> history
     1  ls
     2  cd Link to folderA
     3  ls
     4  cd -P .
     5  cd ..
     6  cd -L ../Link to folderA
     7  cd ..
     8  echo
     9  echo -n
    10  echo -n LOL\n
    11  echo -ne LOL\n
    12  echo -n LOL\n
    13  echo -ne LOL\n
USH (/home/viresh/Shell/)  >>> history -d 3
USH (/home/viresh/Shell/)  >>> history
     1  ls
     2  cd Link to folderA
     3  cd -P .
     4  cd ..
     5  cd -L ../Link to folderA
     6  cd ..
     7  echo
     8  echo -n
     9  echo -n LOL\n
    10  echo -ne LOL\n
    11  echo -n LOL\n
    12  echo -ne LOL\n
    13  history
    14  history -d 3
USH (/home/viresh/Shell/)  >>> history -c
USH (/home/viresh/Shell/)  >>> history
     1  history -c
USH (/home/viresh/Shell/)  >>>
```

Note: Additional / invalid arguments are ignored. Failed commands are also stored in the history. The history from the last closed shell session also carries forward to the new shell session.

2.4 pwd

The echo command outputs the arguments as it is.
Format : *pwd* [$-LP$]
Options available:
-P : Resolve all symlinks and show an absolute path.
-L : (default) Do not resolve any symlinks.

Commands to test pwd:

```
USH (/home/viresh/Shell/)  >>> cd Link to folderA
USH (/home/viresh/Shell/Link to folderA/)  >>> pwd
/home/viresh/Shell/Link to folderA/
USH (/home/viresh/Shell/Link to folderA/)  >>> pwd -P
/home/viresh/Shell/test/folderA
USH (/home/viresh/Shell/Link to folderA/)  >>>
```

Note: Additional / invalid arguments are ignored.

2.5 exit

The exit command exits the shell.
Format : *exit* [$-s$] [*return value*]
Options available:
-S : Suppress the exit greeting.
In case no return value is provided, 0 will be returned.
Commands to test exit:

```
USH (/home/viresh/Shell/)  >>> exit
Thank You !!!
viresh@viresh-PC:~/Shell$


USH (/home/viresh/Shell/)  >>> exit -s
viresh@viresh-PC:~/Shell$
```

Note: Additional / invalid arguments are ignored.

3 External Commands

USH implements the following five external commands:

-> ls Options : -a and -q

-> cat Options : -n and -b

-> date Options : -r and -R

-> rm Options : -d and -i

-> mkdir Options : -p and -m

The code for these commands resides in seperate binary (one for each), and the binary must be located with the USH binary AND have the same name as is used on the shell prompt.

3.1 ls

The ls command lists the folders and directories in the current working directory.
Format : ls $[-aQ]$ $[pathtodirectory]$
Options available:
-a : Show hidden files as well (names starting with .).
-Q : Put quotation marks in the names of the listed files.
-1 : Place each filename on a single line.

Commands to test ls:

```
USH (/home/viresh/Shell/)  >>> ls
bin  Link to folderA  Makefile  src  test
USH (/home/viresh/Shell/)  >>> ls -a
.  ..  bin  Link to folderA  Makefile  src  test
USH (/home/viresh/Shell/)  >>> ls -aQ1
"."
".."
"bin"
"Link to folderA"
"Makefile"
"src"
"test"
USH (/home/viresh/Shell/)  >>> ls -aQ test
"."  ".."  ".sampleTextDoc"  "file1.txt"  "file2.txt"  "folderA"
```

Note: Additional / invalid arguments are ignored. Spaces in path provided are not handled. They will give an error !

3.2 cat

The cat command reads the files provided (in the given order) and prints the concatenated output on the stdout.
In case no input files are provided, it will read from the standard input and print on the standard output (Until a EOF is encountered {^D}) .
Format : ls $[-aQ]$ $[pathtodirectory]$
Options available:
-n : Number all lines.
-b : Do not number the newlines.

Commands to test cat:

```
USH (/home/viresh/Shell/test/)  >>> cat file1.txt
This is some text

coming from file 1

USH (/home/viresh/Shell/test/)  >>> cat -n file1.txt file2.txt
     1  This is some text
     2
     3  coming from file 1
     4
```

```
     5  This is some more text
     6
     7  and still useless
     8  but this time
     9  it's coming from file 2
    10
USH (/home/viresh/Shell/test/)  >>> cat -b file2.txt
     1  This is some more text

     2  and still useless
     3  but this time
     4  it's coming from file 2

USH (/home/viresh/Shell/test/)  >>>
```

Note: Additional / invalid arguments are ignored. In case a directory name is provided to cat, cat will simply ignore the directory. If a file is not open-able, cat will throw an error.

3.3 date

The date command prints the current system date in a human readable format.
Format : $date\ [-rR]\ [pathtofile/folder]$
Options available:
-r <path> : Print the last modified date of given path, throw an error if path invalid.
-R : Output date in RTC format (Sun, 21 Jan 2018 11:18:28 +0530).
Default format is : Sun Jan 21 11:18:31 IST 2018.
Commands to test date:

```
USH (/home/viresh/Shell/)  >>> date
Tue Jan 23 00:05:56 IST 2018
USH (/home/viresh/Shell/)  >>> date -R
Tue, 23 Jan 2018 00:06:15 +0530
USH (/home/viresh/Shell/)  >>> date -r test/file1.txt -R
Tue, 23 Jan 2018 00:00:03 +0530
```

Note: Additional / invalid arguments are ignored. Spaces in path provided are not handled. They will give an error !

3.4 rm

The rm command removes the given list of files (and empty directories if provided -d).
Format : $rm\ [-di]\ < file1 >< file2 >< file3 >$
Options available:
-i : Ask for confirmation before removing anything.
-d : Remove empty directories.

Commands to test rm:

```
USH (/home/viresh/Shell/)  >>> cd test
USH (/home/viresh/Shell/test/)  >>> ls -a
.  ..  .sampleTextDoc  file1.txt  file2.txt  folderA
USH (/home/viresh/Shell/test/)  >>> rm -i .sampleTextDoc
```

```
Do you want to remove .sampleTextDoc ? (y / n)
y
USH (/home/viresh/Shell/test/)  >>> ls -a
.  ..  file1.txt  file2.txt  folderA
USH (/home/viresh/Shell/test/)  >>> rm -i folderA
Do you want to remove folderA ? (y / n)
n
USH (/home/viresh/Shell/test/)  >>> rm folderA
Error Occurred while attempting to remove folderA
USH (/home/viresh/Shell/test/)  >>> ls folderA
emptyFile.txt
USH (/home/viresh/Shell/test/)  >>>
```

Note: Additional / invalid arguments are ignored. Spaces in filename are not handled. Use them carefully, you might end up deleted some un-intended file. If no argument is provided, an error is shown.

3.5 mkkdir

The mkdir command creates a folder .
Format : $mkdir\ [-mp]\ <pathoffoldertomake>$
Options available:
-m : sets the permissions on the created folder (similar to chmod, enter a octal like 777).
-p : create the parent directories as well if the do not exist.
Default invocation is equivalent to mkdir -m 777 <path>.

Commands to test mkdir:

```
USH (/home/viresh/Shell/)  >>> ls
bin  Link to folderA  Makefile  src  test
USH (/home/viresh/Shell/)  >>> mkdir newDir
USH (/home/viresh/Shell/)  >>> ls
bin  Link to folderA  Makefile  newDir  src  test
USH (/home/viresh/Shell/)  >>> mkdir newDir/testdir wow/parent dir2
mkdir: Cannot create the directory wow/parent !
USH (/home/viresh/Shell/)  >>> ls
bin  dir2  Link to folderA  Makefile  newDir  src  test
USH (/home/viresh/Shell/)  >>> mkdir -m 722 -p wow/parent
USH (/home/viresh/Shell/)  >>> ls
bin  dir2  Link to folderA  Makefile  newDir  src  test  wow
USH (/home/viresh/Shell/)  >>> ls wow
parent
USH (/home/viresh/Shell/)  >>> Thank You !!!
viresh@viresh-PC:~/Shell$ ll | grep wow
drwx-w----   3 viresh viresh 4096 Jan 21 11:38 wow/
viresh@viresh-PC:~/Shell$
```

Note: Additional / invalid arguments are ignored. Spaces in path provided are not handled. They will give an error !
Also note that if the underlying filesystem is NTFS, then the -m command will not function as expected.

4 Possible Issues

In all the commands, I have used safe input wherever the user is involved (for e.g yes/no prompt in the rm -i), and thus prevented any direct buffer overflows. But the user can create several symbolic links and trick the cd commands into not resolving the symlinks and then causing an indirect buffer overflow (the BUFSIZE is 1000, so it should take some amount of efforts to achieve this).
Also in almost all commands invalid arguments are ignored, so an attacker cannot use this. The history command has a limited buffer size, although it has been safeguarded against overflowing, it might be possible to tamper with the .ush_history file (which loads history of previous shell sessions) and cause an overflow or prevent the shell from loading.

5 Known Bugs

Also it is known that the rm command as provided does not handle spaces in any way, hence the command has no way to differentiate in
rm abc def (here abc and def are two different files)
and
rm ”abc def” (here it is intended that ”abc def” is one file, but it will be treated as if ”abc and def” are two different files).
Also an assumption is that the shell will be run on a linux system that implements current binary path resolution via /proc/self/exe.
Many unix systems do not confirm to this like Solaris etc. This shell will fail completely on this.
Also the echo command has a limitation of a buffer size, an attempt to print a line of length greater than the buffer size will likely cause issues.

6 Extra Features

USH also implements rudimentary support for alias. These aliases are hardcoded currently and there is only one such alias, the ‘ll‘, which is short for ‘ls -a1‘.
Also in order to support more commands, one can simply place the correspoding binary together with the source binary of USH and it will be usable via the exec mechanism in the shell.