Shell

Félix Yvonnet

January 2023

1 Introduction

I tried to keep the execution as easy as possible so I kept the initial launching actions : first call "make" to create the executable then call it by typing "./shell" in your prompt.
However some libraries may ask some installations (such as 'libreadline-dev', 'bison' and 'flex' that where present in the initial code and that I didn't delete).

2 Answers

2.1

To my mind 'execvp' is the best choice since it takes on a vector, which is what we are feeding him *via* 'cmd -> args' and the suffix p means that it uses the path (so have access to all commands that it allows such as 'ls', 'cat'...) so it is also very useful.

2.2

The sequence operator is ";". It behaves differently from the and ("&&") operator since it executes all actions if no error is raised. For example
'$false ; echo sequence || echo and' will print "sequence" while
'$false && echo sequence || echo and' will print "and".

2.3

Just got the job done

2.4

'rm /dev/null — echo b 2>/dev/null' will raise an error but
'(rm /dev/null — echo b) 2>/dev/null' wont.
In fact "2>" operator is read before the other in the tree from the parser. Hence

it will be only evaluated with the previous argument if no parentheses are used. Factually the role of parentheses is to force the associativity.

2.5

Well if not corrected the problem is that ^C is the terminal signal to stop all processes so it will also stop our new bash... To avoid it I called signal(2) which sets the interpretation of an INTerruption SIGnal (SIGINT constant in signal.h) to do the actions described in an inputed handler. I chose as handler something that completely ignore what happen since the signal sent will stop all other action as wanted but print the prompt in a new line to make it look clean. However it makes it print the prompt before errors... sad...

2.6

Since we haven't implemented the redirect function yet, the 'ls > dump' call only prints the ls call but doesn't know what to do with the rest... My first implementation just led to a good functioning of the ls call but didn't do any backup.
To avoid it we have to use dup2(2) to redirect the flux depending on where we want to redirect inside a fork(2) to end redirections at the end of the file execution.

2.7

When we call 'cat main.c | grep main' we are looking in the file "main.c" all occurrences of main. To do so we have to take as input of the "grep main" part the output of the "cat main.c" part. We can do it easily with pipe(2) and multiple fork(2) in order to have a parallel execution of both sides.
Also only dup2 stdout to stdin will just break the thing off since the stdin will wait to the infinite and beyond because stdout may at a moment send a new message.

3 Basic examples

In the previous questions I've already given some examples but here is more (try rewriting them in that order in the shell, don't copy-paste the quotation marks for example don't appeared the same) :

- echo \" # Enable printing of special characters
- ls -l > dump
- cat dump — grep main
- echo -e "\e[35mHello Purple World\e[0m"

- var = MyString # I allowed space between equality sign
- echo $var > test.txt
- grep My < test.txt
- rm /dev/null 2> /dev/null
- echo "Output status: $?"
- name = me && var = 1 && echo "name: $name, var: $var"
- set; unset var
- echo "I'm $name working at $$ pid"
- echo $var # var not defined

4 Features added

‘cd‘

The cd function is the only one that has its place in the shell implementation since all the other (‘ls‘, ‘cat‘) are present in the PATH variable and not in the bash Linux...

I decided to add some features such as the ‘cd -‘ that change to the previous directory or the ‘cd‘ without argument that sends to the home directory.
In addition to make it more readable I added the current working directory in the message of the prompt and added colors to looks like my shell.

Please note that the parser/lexer where unfinished, I had multiple problems with them, for example the fact that they didn't recognize the "-" sign... For more information go to section "Problems". To resolve it I first added (some of) the missing characters, then coded a function that replace the symbol " " with getenv("HOME") (in general /home/user) when it is alone or followed by a slash symbol (I wanted to make it possible to write ‘touch .txt‘ like in bash).

‘cat‘

Well not many things to say here, it just prints what is in file(s) given as argument(s). It can be used with ‘cat dump — grep main‘ which will probably print "main.c\n main.o".
I also added the possibility to print from the stdin, ie I implemented the ‘cat -‘ or ‘cat ‘ command.

‘ls‘

I'm not that proud of that one, I haven't done many argument possibilities. Hence if you use the ‘ls‘ command you will use the execvp(ls). If you want to try mine go to the function int handle_C_PLAIN(struct cmd *cmd) and uncomment the line in the first fork that tests whether the command is "ls" and make it change for my ls. (It will just print elements in the inputted directory.ies.

4.1 variables

II added a fixed-length environment that will contain variables, it's coding is sub-optimal but easier and quicker to code (and almost as fast but more limited in space than a linked list coding). I also added basic variables like ‘$¿ that send the output state of the previous command-line and ‘$$‘ that send the id of the main branch (quite useless here but easy to code). As I wanted to access the PATH variable easily I made it possible to have access to "global" variables defined in the real bash when they are not defined in the source. To create a new global variable call ‘var == 1‘ and ‘gdset var‘ to remove it. You can that way change the global PATH variable. However these variables will be ignored once back to your real shell. Finally I added the ‘set‘ and ‘unset‘ commands that respectively show all variables defined previously and remove a variable from the environment.
In general variables work quite like bash variables but I didn't accepted "$" in variable values in order to avoid bug due to my decoding of variables and asked only for alphanumerical variable names. Moreover it is possible to have space between the equal sign and the rest such as ‘var = hello‘ unlike in bash.
I didn't support evaluation like ‘var = 1; echo $(($var+1))‘ since the right part is not parsed because I didn't wanted to spend weeks on it.
In fact I just look into all strings if I see a $ and replace the following name by its value before working on the command.

5 Problems

I had some problems during this homework. The first one being a big lack of documentation. I spent too much time testing with some printf to understand what was what. For example it is not intuitive that redirections are neither in the args nor in the right but in the output. To make it more quick to understand maybe a quick paragraph at the beginning explaining the tree structure of cmd and where are stored data could be helpful.
But the main problem for me ¡as the parser and lexer. Though we were not supposed to change them (regardless of the bonuses), there was also many characters left, such as " " and most of non alphanumerical symbols. Even the minus sign "-" was not available (since it's a special character for the lexer so you should add a \ before it). It led to a huge misunderstanding of problems...
To handle this problem it may be interesting to dive into the parser and lexer

in order to add special characters. It can be done quickly and will help many people.

6 Conclusion

Well thanks for reading and have a lovely day :)