

CS416 Homework 3 – shell.c

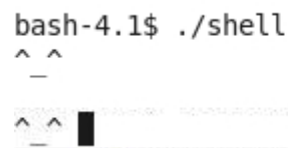
Team members: Zhenhua Jia, Yimeng Li, Yihan Qian

General notes:

Our code has successfully passed all the testing requirements. This reports offers screenshots corresponding to all the requirements. Some of the testing results have been combined into one figure due to the result similarity.

1. Empty command #1

Test: enter just a newline. All that should happen is that you should get another prompt.



```
bash-4.1$ ./shell
^ ^
_
```

Figure 1

2. Empty command #2

Test: enter a bunch of spaces and tabs followed by a newline. All that should happen is that you should get another prompt.



```
bash-4.1$ ./shell
^ ^
_
```

Figure 2

3. Exit on end of file

Enter a control D at the command prompt. The shell should exit cleanly (just like with the *exit* command).



```
bash-4.1$ ./shell
^ ^
_
```

Figure 3

4. A shell prompt for each line of input

The shell should issue a prompt. The prompt should NOT be on a separate line.

```

^ ^ aaaa          aaaaaaa          aaaaa          aaaa  a
_ _ aaaaaaaaaa

```

Figure 4

5. No shell prompt if the standard input is not a terminal

Your shell should not print a prompt if the input is not a terminal. For example, if you run:

`(echo echo abc;echo echo def;echo echo ghi)|./myshell`

```

bash-4.1$ (echo echo abc;echo echo def;echo echo ghi)|./shell
Command 1 is :echo abc

abc
process 12129 exits with 0

Command 1 is :echo def

def
process 12130 exits with 0

Command 1 is :echo ghi

ghi
process 12131 exits with 0

bash-4.1$ 

```

Figure 5

6. The shell should run a simple command with arguments.

The shell should print an exit code for each process that terminates.

If you run:

`echo hello, worl:`

```

^ ^ echo hello,world
_ _ Command 1 is :echo hello,world

hello,world
process 15996 exits with 0

^ ^ 
_ _ 

```

Figure 6

7. Handle too many arguments gracefully

Your shell should warn of too many arguments on a command line and definitely not crash. This command, for example, should be handled gracefully:

```
echo a b c d e f g h i j k l m n o p q r s t u v w x y z a b c d e f g h i j k l m n o p q r s t
u v w x y z a b c d e f g h i j k l m n o p q r s t u v w x y z a b c d e f g h i j k l m n o p
q r s t u v w x y z
```

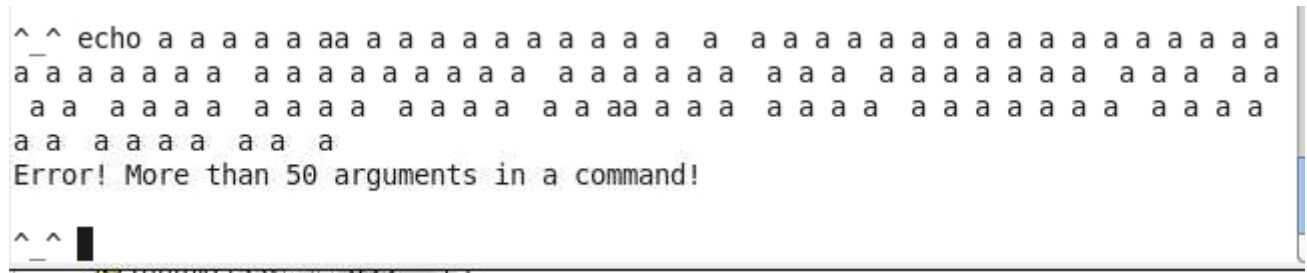


Figure 7

8. Quotes should be handled properly

If you run:

```
echo "abc" "def" 'ghi' 'jkl'
```

A terminal window showing the command 'echo "abc" "def" 'ghi' 'jkl'' being executed. The output is 'abc def ghi jkl'. Below the output, it says 'process 20751 exits with 0'. The prompt is '^ _'.

Figure 8

9. Spaces within quotes should be parsed correctly.

Download and compile [echoargs.c](#) (from the [getopt tutorial](#)). Run

```
./echoargs 'abc def ghi' "j      k    l"
```

```

bash-4.1$ ./echoargs 'abc def ghi' "j k l"
bash: ./echoargs: No such file or directory
bash-4.1$ cc -o echoargs echoargs.c
bash-4.1$ ./echoargs 'abc def ghi' "j k l"
argc = 3
arg[0] = "./echoargs"
arg[1] = "abc def ghi"
arg[2] = "j k l"
bash-4.1$ █

```

Figure 9

10. Mismatched quotes should be detected and handled gracefully #1.

For example, running:

echo 'abc

```

_
^ _ echo 'abc
Error. No sencond ' mark found.
^ _ █

```

Figure 10

11. Mismatched quotes should be detected and handled gracefully #2.

For example, running:

echo 'abc"

```

_
^ _ echo 'abc"
Error. No sencond ' mark found.
^ _ █

```

Figure 11

12. A command should work with a single pipe.

ls -laF |tr a-z A-Z

```

^ ^ ls -laF / | tr a-z A-Z
Command 1 is :ls -laF /

Command 2 is :tr a-z A-Z

TOTAL 2548
DR-XR-XR-X.    36 ROOT ROOT    4096 MAR  2 01:33 ./
DR-XR-XR-X.    36 ROOT ROOT    4096 MAR  2 01:33 ../
DRWXR-XR-X.     9 ROOT ROOT    4096 AUG 22 2013 .AUTofs/
-RW-R--R--     1 ROOT ROOT         0 FEB 28 19:08 .AUTofsck
-RW-R--R--     1 ROOT ROOT         0 AUG 23 2012 .AUTORELABEL
DR-XR-XR-X.     2 ROOT ROOT    4096 FEB 21 03:53 BIN/
DR-XR-XR-X.     5 ROOT ROOT    4096 DEC  3 05:26 BOOT/
DRWXR-XR-X.     2 ROOT ROOT    4096 DEC  9 03:36 CGROUP/
DRWXR-XR-X.    20 ROOT ROOT   4320 MAR  2 17:08 DEV/
DRWXR-XR-X.   203 ROOT ROOT  20480 MAR  2 21:39 ETC/
DRWXR-XR-X.     3 ROOT ROOT    4096 SEP 13 2012 FILER/
DRWXR-XR-X.     2 ROOT ROOT    4096 AUG 22 2013 FREESPACE/
DRWXRWXRWT.    43 ROOT ROOT    4096 FEB 24 01:34 .FREESPACE/
DRWXR-XR-X.     2 ROOT ROOT    4096 AUG 21 2012 HOME/
DRWXR-XR-X.     2 ROOT ROOT    4096 SEP 13 2012 ILAB/
DRWX-----     3 ROOT ROOT    4096 NOV  7 2012 .KDE/
DRWXR-XR-X.     2 ROOT ROOT    4096 APR 17 2013 KOKO/

```

Figure 12

13. A pipeline of three commands should work.

This pipeline of commands:

ls -alF / | grep bin | cat -n

```

^ ^ ls -alF / | grep bin | cat -n
Command 1 is :ls -alF /

Command 2 is :grep bin

Command 3 is :cat -n

    1 dr-xr-xr-x.    2 root root    4096 Feb 21 03:53 bin/
    2 dr-xr-xr-x.    2 root root   12288 Feb 26 03:29/sbin/
process 29548 exits with 0
process 29549 exits with 0
process 29550 exits with 0

^ ^ █

```

Figure 13

14. A pipeline of more than three commands should work.

This command:

```
ls -alF / | grep bin | tr a-z 'A-Z' | rev | cat -n
```

```
^_ ^ ls -alF / | grep bin | tr a-z 'A-Z' | rev | cat -n
```

```
Command 1 is :ls -alF /
```

```
Command 2 is :grep bin
```

```
Command 3 is :tr a-z A-Z
```

```
Command 4 is :rev
```

```
Command 5 is :cat -n
```

```
process 1474 exits with 0
```

```
process 1475 exits with 0
```

```
process 1476 exits with 0
```

```
1 /NIB 35:30 12 BEF 6904 T00R T00R 2 .X-RX-RX-RD
```

```
2 /NIBS 92:30 62 BEF 88221 T00R T00R 2 .X-RX-RX-RD
```

```
process 1477 exits with 0
```

```
process 1478 exits with 0
```

```
^_ ^ █
```

Figure 14

15. The cd command should work with one argument

The cd command should work with no arguments

```
^_^ cd
Command 1 is :cd

^_^ pwd
Command 1 is :pwd

/autofs/ilab/ilab_users/yq40
process 6501 exits with 0

^_^ cd /Documents
Command 1 is :cd /Documents

^_^ pwd
Command 1 is :pwd

/autofs/ilab/ilab_users/yq40/Documents
process 6874 exits with 0

^_^ cd /etc
Command 1 is :cd /etc

cd: No such file or directory: /etc

^_^ pwd
Command 1 is :pwd

/autofs/ilab/ilab_users/yq40/Documents
process 10006 exits with 0

^_^
```

Figure 15

16. The cd command should barf when given more than one argument

Running the cd command with two or more arguments, such as:

```
^_^ cd /etc /home
Command 1 is :cd /etc /home

cd error: Too many arguments!

^_^
```

Figure 16

17. The exit command should exit with no arguments

Run exit. The shell should exit. Run echo \$0 (\$? is the environment variable that contains the exit code of the last command) and you should see 0, representing an exit code of 0.

```

^ ^ exit
_ Command 1 is :exit

bash-4.1$ echo $?
0
bash-4.1$ █

```

Figure 17

18. The exit command should exit with an argument

Run exit 123. The shell should exit. Run echo \$0 (\$? is the environment variable that contains the exit code of the last command) and you should see 123, representing an exit code of 123.

```

^ ^ exit 125
_ Command 1 is :exit 125

bash-4.1$ echo $?
125
bash-4.1$ █

```

Figure 18

19. Check that the shell is able to accept at least 10 commands.

Run this in bash, replacing *myshell* with the name of your shell:

(typeset -i i;i=0;while [\$i -lt 10];do echo echo \$i;i=i+1;done)|./myshell


```
bash-4.1$ (typeset -i i;i=0;while [ $i -lt 10 ];do echo echo $i;i=i+1;done)|./shell
Command 1 is :echo 0
0
process 14452 exits with 0

Command 1 is :echo 1
1
process 14453 exits with 0

Command 1 is :echo 2
2
process 14454 exits with 0

Command 1 is :echo 3
3
process 14455 exits with 0

Command 1 is :echo 4
4
process 14456 exits with 0

Command 1 is :echo 5
5
process 14457 exits with 0
```

Figure 19

```
4
process 14456 exits with 0

Command 1 is :echo 5

5
process 14457 exits with 0

Command 1 is :echo 6

6
process 14458 exits with 0

Command 1 is :echo 7

7
process 14459 exits with 0

Command 1 is :echo 8

8
process 14460 exits with 0

Command 1 is :echo 9

9
process 14461 exits with 0

bash-4.1$ █
```

Figure 20

22. Check that the shell is able to accept at least 1000 commands.

Run this in bash, replacing *myshell* with the name of your shell:

```
(typeset -i i;i=0;while [ $i -lt 1000 ];do echo echo $i;i=i+1;done)|./myshell
```

```
992
process 20133 exits with 0

Command 1 is :echo 993

993
process 20134 exits with 0

Command 1 is :echo 994

994
process 20135 exits with 0

Command 1 is :echo 995

995
process 20136 exits with 0

Command 1 is :echo 996

996
process 20137 exits with 0

Command 1 is :echo 997

997
process 20138 exits with 0

Command 1 is :echo 998

998
process 20139 exits with 0

Command 1 is :echo 999

999
process 20140 exits with 0

bash-4.1$ █
```

Figure 21