

Your Own Linux Shell

Due Date: 3/19/20 at 11:59pm

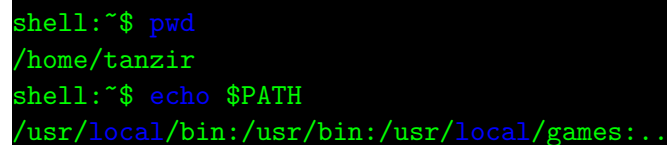
Introduction

Most useful interaction with a UNIX/Linux-based system occurs through the shell. Using a series of easy to remember and simple commands, one can navigate the UNIX file system and issue commands to perform a wide variety of tasks. Even though it may appear simple, the shell encapsulates many significant components of the operating system.

Basic Shell Features

Environment

The shell maintains many variables which allow the user to maintain settings and easily navigate the filesystem. Two of these that are particularly important are the current working directory and the PATH. As its name implies, the current working directory variable keeps track of the user's current directory. The PATH variable consists of string of colon separated pathnames. Whenever you type a name of a command, the kernel searches in the directories specified by the PATH variable starting with the leftmost directory first. If the executable is not found in any of the specified directories, then the shell returns with an error. One may modify the PATH at any time to add and remove directories to search for executables.



```
shell:~$ pwd
/home/tanzir
shell:~$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/games:..
```

Figure 1: Current directory and the PATH enviromental variable.

Pipelining

UNIX provides a variety of useful programs for you to use (grep, ls, echo, to name a few). Like instructions in C++, these programs tend to be quite effective at doing one specific thing (Such as grep searching text, ls printing directories, and echo printing text to the console). However, programmers/OS users would like to accomplish large tasks consisting of many individual operations. Doing such requires using results from previous steps in order to complete a larger problem. The UNIX shell supports this through the pipe operation (represented by the character |). A pipe in between two commands causes the standard output of one to be redirected into the standard input of another. An example of this is provided below, using the pipe operation to search for all processes with the name "bash".

```
shell:~$ ps -elf | grep bash | awk '{print $10}' | sort
3701
4197
```

Figure 2: Pipelining multiple commands.

Input/Output Redirection

Many times, the output of a program is not intended for immediate human consumption (if at all). Even if someone isn't intending to look at the output of your program, it is still immensely helpful to have it print out status/logging messages during execution. If something goes wrong, those messages can be reviewed to help pinpoint bugs. Since it is impractical to have all messages from all system programs print out to a screen to be reviewed at a later date, sending that data to a file as it is printed is desired.

Other times, a program might require an extensive list of input commands. It would be an unnecessary waste of programmer time to have to sit and type them out individually. Instead, pre-written text in a file can be redirected to serve as the input of the program as if it were entered in the terminal window.

In short, the shell implements input redirection by redirecting the standard input of a program to an file opened for reading. Similarly, output redirection is implemented by changing the standard output (and sometimes also standard error) to point to a file opened for writing.

```
shell:~$ echo "This text will go to a file" > temp.txt
shell:~$ cat temp.txt
This text will go to a file
shell:~$ cat < temp1.txt
This text came from a file
```

Figure 3: I/O redirection.

Assignment

For this assignment, you are to design a simple shell which implements a subset of the functionality of the Bourne Again Shell (Bash). The requirements for your shell are as follows:

- Continually prompt for textual user input on a command line.
- Parse user input according the provided grammar (see below)
- When a user enters a well formed command, execute it in the same way as a shell. You must use the commands fork and exec to accomplish this. You may NOT use the C++ system() command.
- Allow users to pipe the standard output from one command to the input of another an arbitrary number of times.

- Support input redirection from a file and output redirection to a file.
- Allow users to specify whether the process will run in the background or foreground using an '&'. Backgrounding processes should not result in the creation of zombie processes. (Commands to run in the foreground do not have an '&', and commands that run in the background do)
- Print a custom prompt which supports printing the current directory, user name, current date, and current time.
- (Bonus) Allowing \$-sign expansion. See the last sample command in the list of commands for example.

Write a report describing your design for piping, redirection and other special techniques that you are unique to your implementation. No need to restate the obvious.

Figure #4: Simple Shell Grammar

```
valid_string = unix_command || unix_command AMP || special_command

unix_command = command_name ARGS || unix_command REDIRECTION filename ||
               unix_command PIPE unix_command

special_command = cd DIRECTORY || exit

command_name = any valid executable/interpreted file name

AMP = &

ARG = string

ARGS = ARG ARGS || ARG

DIRECTORY = absolute path || relative path

PIPE = |

REDIRECTION = < || >
```

Figure #4: Some Sample Commands

```
echo Here starts our shell ...
echo -e "<<<<< This message contains a line feed >>>>>\n"
echo "<<<<< Start to exercise pipe >>>>>"

echo "<<<<< IO redirection >>>>>"
ps > test.txt
```

```

grep pts < test.txt
pwd > pwd.txt
mv pwd.txt newpwd.txt
cat newpwd.txt

echo "<<<<< 1 pipe >>>>>"
ps -a | head -5
echo "<<<<< 2 pipes >>>>>"
ps -a | awk '/pts/[0-9]/{print $1}' | tail -5
ps -a | awk '/pts/[0-9]/{print $2}' | tail -2

echo "<<<<< 3 pipes >>>>>"
ls -l /proc/sys | awk '{print $9}' | sort -r | head -5
ls -l /proc/sys | awk '{print $8}' | head -3 | sort -r
ls -l /proc/sys | awk '{print $8$9}' | head -10 | sort

echo "<<<<< 4 pipes with I/O redirection >>>>>"
ls -l /proc/sys > test.txt
awk '{print $8$9}' < test.txt | head -10 | head -8 | head -7 | sort > output.txt
cat output.txt

echo "<<<<< Background process (ps after some time to make sure the bg's are not
defunct) >>>>>"
dd if=/dev/zero of=/dev/null bs=1024 count=10485760 &
sleep 10 &
ps

echo "<<<<< Directory Operations >>>>>"
cd /home/ugrads/
pwd
cd -
pwd

echo "<<<<< Miscellenous >>>>>"
rm newpwd.txt
jobs
sleep 10

echo "<<<<< Bonus ($-sign expansion) >>>>>"
cat /proc/$(ps|grep bash|head -1|awk '{print $1}')/status

```