Student Information

Liam McFadden (liamm18)
<REDACTED>

How to execute the shell

The shell is executed by typing 'cush' into your command line.

Important Notes

This implementation of 'cush' is relatively standard. Commands are issued by typing the name of whatever command you want to run and pressing enter. Pipes, i/o redirection, running programs that require exclusive access to the terminal, and redirecting stderr are all supported. Jobs are controlled via the given job_list struct and a list of job/pid pairs for fetching a job a given a certain pid.

Description of Base Functionality

jobs - This command was implemented by simply looping through the "job_list" and calling "print_job(job)" for each job in the list. To ensure that the list remains up to date, a function called "update_jobs" is issued throughout the main read/eval loop. More specifically, the function is called at the beginning of the pipeline loop, after a new job/pid pair is created, and whenever a pipeline is removed from the "pipes" list.

fg - This command was implemented by sending SIGCONT to whichever jid was specified from the command-line. If the jid does not exist or was not provided, a corresponding error message is printed. If the corresponding job's status is "STOPPED" then we know that there is a termstate to be restored so we give the termstate back to that job's "saved_tty_state". Once the termstate has been restored (if necessary) and given back to jid, we set the job's status to FOREGROUND.

bg - This command was implemented similarly to fg, except the shell remains in control of the termstate. The provided jid is still sent SIGCONT, and if no jid or an invalid jid is provided, an error message is printed. Once the job has been resumed, it's status is set to BACKGROUND.

kill - This command's implementation is rather simple. After making sure a valid jid is provided, and outputting an error if otherwise, "killpg" is called with the signal SIGTERM. The job corresponding to jid's pgid is set to be the target. If the "killpg" call returns -1, then an error message saying that the job was not found is printed.

stop - Stop is nearly identical to kill, except for two things: the call to "killpg" sends SIGSTOP and the job's status is set to STOPPED at the end. Other than those two things, the error checking and format of the implementation are the same.

\^C - Ctrl C is implemented via properly controlling jobs/process groups and ignoring the signal it sends (SIGINT) while the prompt is being built. This ensures that only the foreground process group (excluding the shell) will receive the signal, thus preventing a user from exiting the shell via a ^C invocation.

\^Z - Ctrl Z is implemented via proper job/process group control and by catching the signal sent by ctrl Z (SIGSTOP). The signal is caught in the "handle_child_status" function via the WIFSTOPPED macro. Once the signal has been detected, the termstate is saved to the corresponding job struct, the status is set to STOPPED, the termstate is given back to the shell, and the job is printed to the terminal.

Description of Advanced Functionality

I/O - For redirecting the output of a command, we check if the current command is the last in the pipeline and if the command has a file to redirect it's output to. If so, we attempt to open the specified file in the create and write only modes. If unsuccessful, an error message is printed and the command writes to stdout instead. Otherwise, stdout is closed, and "dup" is called with the new file descriptor as its parameter. If "dup" fails, an error message is displayed. If the command wants to redirect stderr as well, stderr is closed and an additional call to "dup" is given with the specified file descriptor as its parameter. If the command wants to redirect output in "append mode", the specified file is opened in the create, write-only, and append modes.

For redirecting the input of a command, we check if the pipeline specifies a file for input redirection and if we are on the first command. If so, we open the file in read-only mode and call "dup2" with the specified file's fd as the first parameter and STDIN_FILENO as the second parameter. If either "open" or "dup2" return -1, an error message is printed.

Pipes - This implementation uses two pipes. Any pipeline greater than 2 commands uses both. A variable named "i" (starting at 0) is used to keep track of which number command the loop is currently at. This is useful because the pipes behave differently on the first and the last commands.
On the first command, we point stdout to the write end of pipe 1 and close pipe1. If i is odd and it is the last command, we point the read end of pipe1 to stdin and close pipe1. If i is even and it is the last command we point stdin to the read end of pipe2 and then close pipe2. Note: any "pointing" of stdout/stdin to the write/read end of a pipe is done with dup2, where pipe1/2 is the first argument and the stdin/stdout fd is the second argument. For commands sandwiched between two other commands where i is odd, we point stdout to the write end of pipe2 and stdin to the read end of pipe1. We then close both pipes. For the same case where i is even, we point stdout to the write end of pipe1 and stdin to the read end of pipe2. We then close both pipes. In the parent process we close pipe1 when i is odd and pipe2 when i is even. For stderr redirection, we check if "dup_stderr_to_stdout" is true for all cases except for the last command. If it's true, then we point stderr to the write end of whatever pipe stdout is currently writing to.

Exclusive

```
Access- Exclusive access to the terminal was implemented through
propper job control and termstate
        preservation. This allows for programs like vim to regain
exclusive access if
        they're resumed after being stopped.

List of Additional Builtins Implemented
---------------------------------------
cd -    Basic change directory functionality. Most of the features
you'd expect in bash work here.
        Calling "cd" by itself takes you to your home directory, "cd
~" takes you to your home
        directory, and "cd ~/some/other/path" is functional. If a user
attempts to cd to a
        non-existent directory or restricted directory, an error
message is printed.

custom
prompt- The prompt displays basic information, such as current user,
branch of rlogin that is in use,
        and the current folder the user is in.
```