Creating an interactive user defined shell Part -1

The goal of the project is to create a user defined interactive shell program that can create and manage new processes. The shell should be able to create a process out of a system program like emacs, vi or any user-defined executable

The following are the specifications for the project. For each of the requirement an appropriate example is given along with it.

Specification 1: Display requirement

When you execute your code a shell prompt of the following form must appear:
<username@system_name:curr_dir>
E.g., <Name@UBUNTU:~>

The directory from which the shell is invoked will be the home directory of the shell and should be indicated by "~" If the user executes "cd" change dir then the corresponding change must be reflected in the shell as well.
E.g., ./a.out
<Name@UBUNTU:~>cd newdir
<Name@UBUNTU:~/newdir>

Specification 2: Builtin commands

Builtin commands are contained within the shell itself. Checkout 'type *commandname*' in the terminal (eg. 'type echo') When the name of a builtin command is used as the first word of a simple command the shell executes the command directly, without invoking another program. Builtin commands are necessary to implement functionality impossible or inconvenient to obtain with separate utilities.

Make sure you implement **cd**, **pwd** and **echo**.

Specification 3: ls command

Implement **ls [al]** – (it should be able to handle ls, ls -l, ls -a and ls -al/la. For ls and ls -a, outputting the entries in a single column is fine.

Specification 4: System commands with and without arguments

All other commands are treated as system commands like : emacs, vi and so on. The shell must be able to execute them either in the background or in the foreground.

Foreground processes*: For example, executing a "vi" command in the foreground implies that your shell will wait for this process to complete and regain control when this process exits.*

Background processes*: Any command invoked with "&" is treated as background command. This implies that your shell will spawn that process and doesn't wait for the process to exit. It will keep taking user commands.*

E.g
<Name@UBUNTU:~> ls &
This command when finished, should print its result to stdout.

```
<Name@UBUNTU:~>emacs &
<Name@UBUNTU:~> ls -l -a ( Make sure all the given flags are executed properly for any command and
not just ls.)

        .
        .
        . Execute other commands
        .
        .

<Name@UBUNTU:~> echo hello
```

Specification 5: pinfo command (user defined)

-pinfo : prints the process related info of your shell program.

```
<Name@UBUNTU:~>pinfo
pid -- 231
Process Status -- {R/S/S+/Z} memory
- 67854      {Virtual
Memory}
Executable   Path   --
~/a.out
```

-pinfo <pid> : prints the process info about given pid.

```
<Name@UBUNTU:~>pinfo 7777
```

Bonus:

If the background process exits then the shell must display the appropriate message to the user.

For example :-

After emacs exits, your shell program should check the exit status of emacs and print it on stderr <Name@UBUNTU:~>

emacs with pid 456 exited normally <Name@UBUNTU:~>

Bonus: User defined commands

-nightswatch [options] <command>: (pun intended) Look up the man page entry for the 'watch' command - 'man watch'.
You will be implementing a modified, very specific version of watch. It executes the command until the '**q**' key is pressed.

***Options*:**
-n seconds: The time interval in which to execute the command (periodically)

***Command*:**
Either of these two:

1) interrupt – print the number of times the CPU(s) has(ve) been interrupted by the **keyboard controller (i8042 with IRQ 1)**. There will be a line output to stdout once in every time interval that was specified using -n. If your processor has 4 cores (quadcore machine), it probably has 8 threads and for each thread, output the number of times that particular CPU has been interrupted by the keyboard controller. Eg.

```
<Name@UBUNTU:~> nightswatch -n 5 interrupt
CPU0  CPU1  CPU2  CPU3  CPU4  CPU5  CPU6  CPU7
    2     13     2      1      0      2      1      0
    2     13     4      1      0      4      1      0
...
...
...
```

A line every 5 seconds until '**q**' is pressed.

2) dirty – print the size of the part of the memory which is **dirty**. Eg.

```
<Name@UBUNTU:~> nightswatch -n 1 dirty
968 kB
1033 kB
57 kB
...
```

A line every 1 second until '**q**' is pressed.