Evaluation Guidelines
Operating Systems and Networks

Extending the Shell

General Guidelines:

1. Outputs to some commands may vary based on their implementation. While the most common among these variations are listed below, a particular variation should be awarded its share of marks if it is logically sound and produces output it can explain, even if it isn't in the list.

2. Use kill -9 to clean up background processes between tests for different specifications.

Before Evaluating

Run these commands:

```
echo "This is the\ninput text." > in.txt
seq 1 10 > num.txt
chmod 777 num.txt
```

Specification 1: Input/Output Redirection [16]

1. Simple input redirection [2]
   ```
   rev < in.txt
   ```

2. Simple output redirection [2]
   ```
   echo hi > a.txt # a.txt should not previously exist
   cat a.txt
   ```

3. Appending to the output file [1]
   ```
   seq 11 20 >> num.txt
   cat num.txt # num.txt should have numbers from 1 to 20
   ```

4. Overwriting the output file [1]

```
echo so wtf > in.txt
cat in.txt
```

5. Permissions of output files [2]
   `ls -l`

   If the permissions of `a.txt`, `num.txt` and `in.txt` in (1), (2), and (3) don't match, only deduct marks here, and not anywhere else.
   `a.txt` should be `rw-r--r--` [1]
   `num.txt` should be `rwxrwxrwx` and `in.txt` should be `rw-r--r--` [1]

6. Input and output redirection [2]
   ```
   sort -nr < num.txt > out.txt
   cat out.txt # should contain numbers from 20 to 1
   ```

   If this works in any one order (that is accepted by bash), award them full marks.

7. Order of arguments [2]
   ```
   sort -nr < num.txt > out.txt
   sort -nr > out.txt < num.txt

   sort -n < num.txt -r > out.txt
   sort > out.txt < num.txt -n -r
   sort >> out.txt -r < num.txt -n

   # in all cases except the last, out.txt should contain numbers
   from 20 to 1. Delete out.txt before running the next command.
   ```

   Award [1] for the first two and another [1] for the last three.
   Do not test with multiple redirections of the same kind.

8. For background processes [2]
   ```
   rev < in.txt > out.txt &
   cat out.txt # should contain "ftw os" (without quotes)
   ```

   Check for indications of it running in the background: the alert that the process exited should appear.

9. Error handling [2]

```
rev < not.txt # should give an error
echo halo >> not.txt # should not give an error and create the file
```

Award [1] each for the two commands.

Specification 2: Command Pipelines [10]

10. Basic functionality [6, (2 each)]

```
rev num.txt | grep 21 # should print 21
rev num.txt | rev | grep 12 # should print 12
sleep 2 | rev num.txt | rev | grep 12 # should print 12 after 2 seconds
```

11. Error handling [2]

```
ls | wc | bullshit # should print just the error, and not the output of any other command
bullshit | ls # should print an error as well as the output of ls
```

Specification 3: I/O Redirection with Command Pipelines [20]

12. Basic functionality [15, (3 each)]

```
cat < num.txt | wc -l # should output 20
echo hello | wc -c > out.txt # out.txt should contain 5
cat < num.txt | wc -l > out.txt # out.txt should contain 20
cat < num.txt | wc -l >> out.txt # out.txt should contain 20 twice
grep 12 < num.txt | rev | grep 21 | rev > out.txt # out.txt should contain 12
```

13. Error handling and empty files [5]

```
ls | wc -l > out.txt | cat # out.txt should contain the number of lines in ls, cat should not wait for input and should print nothing to the screen
echo hello | wc -l < num.txt # should print 20
cat < nope.txt | wc -c # should print file not found error and print a 0
```

Specification 4.1 & 4.2: setenv and unsetenv [4]

14. Basic commands [2]

```
setenv foo bar
env | grep foo # should print foo=bar

setenv foo
env | grep foo # should print foo=

unsetenv foo
env | grep foo # should print nothing
```

15. Error handling [2]

```
setenv
setenv foo bar boo
unsetenv
unsetenv blah blah
# all of these trigger an error message
```

Specification 4.3 & 4.7: jobs and overkill [9]

16. jobs [5]

```
sleep 100 &
sleep 100 &
sleep 100
CTRL+Z
jobs
```

Expected output:

```
[1] Running sleep 100 & [<pid>]
[2] Running sleep 100 & [<pid>]
[3] Stopped sleep 100 [<pid>]
```

Check for state, program name (whether it's just `sleep` or `sleep 100` or `sleep 100 &` does not matter) and process ID.

Kill the last process, and again run:
`sleep 100 &`
`jobs`

Expected output:
`[1] Running sleep 100 & [<pid>]`
`[2] Running sleep 100 & [<pid>]` *`# these two should be the same`*
`[3] Running sleep 100 & [<pid>]` *`# this should change`*

The numbering format doesn't matter (if it restarts or not), as long as it is consistent with whatever they say is their numbering format.

If you're not using a virtual machine, you can use `gedit` instead of `sleep`. Do not use `vim` or `vi` or `emacs -nw`.

Note: *If they haven't implemented signal handling (**`CTRL+Z`** doesn't work or affects the entire shell), try to replicate this by running the command in the background and then sending **`SIGTSTP`** using kill, as follows:*
`sleep 100 &`
`ps` *`# note the pid`*
`kill -20 <pid>`
*This should suspend **`sleep`** in the background, and you can test **`jobs`** now.*
If possible to test alternatively, try not to penalise them here for other functionality they haven't implemented.

17. Overkill [4]
    With at least 5 background processes running,
    `overkill`
    `jobs`
    All of them should be killed, and you should see 5 alerts. Note that if the signals aren't spaced out, they can be skipped, so do count the number of alerts. Deduct [2] if it misses some processes.

Specification 4.4: kjob [3]

18. Sending the correct signal [2]

```
sleep 100
CTRL+Z
jobs # sleep should be stopped
kjob <job number> 18 # not pid
jobs # sleep should be running in the background
kjob <job number> 9 # not pid
jobs # sleep should end and not appear, and should show an
alert
```

19. Sending to the correct process [1]

```
sleep 1000 &
sleep 1000
CTRL+Z
jobs # 2nd sleep should be stopped
kjob <job number> 9 # of the second sleep
jobs # 2nd sleep should not appear, and should show an alert,
first sleep should not be affected
```

Note: *Again, use the same trick if* ***CTRL+Z*** *doesn't work.Try to use* ***ps aux*** *if* ***jobs*** *doesn't work.*

Specification 4.5 & 4.6: fg and bg [12]

20. Background [6]

```
sleep 10
CTRL+Z # again, replicate this as above if CTRL+Z does not work
jobs # should be stopped in the background, note job number
bg <job number>
jobs # should be running in the background, and print an alert
on exiting
jobs # after sleep finishes, this should show nothing
```

21. Foreground [6]

```
sleep 10
CTRL+Z # again, replicate this as above if CTRL+Z does not work
jobs # should be stopped in the background, note job number
fg <job number> # the process should come to the foreground,
and be running (i.e, exit after a while) and should not print
an alert on exiting
jobs # after sleep finishes, this should show nothing
```

Deduct [1] from either case if the alert appears (or does not appear) when it shouldn't (or should). Deduct [1] from either case if `jobs` still lists the process when it shouldn't. Award 0 for case 22 if the process is stopped in the foreground (i.e, never finishes).

Specification 4.8: quit [2]

22. Quit [1]

```
quit # should exit the shell
```

23. EOF [1]

On an empty prompt, press `CTRL+D` and it should exit the shell.

Specification 5: Signal handling [9]

24. SIGINT [3]

```
sleep 100
CTRL+C # should interrupt that process
jobs # should not show anything
ps # should also not show sleep
```

25. The shell itself is unaffected [3]

```
CTRL+C
CTRL+Z
```

Nothing should affect the shell.

26. SIGTSTP and background [3]

```
sleep 100
CTRL+Z # should send it to the background
jobs # the process should be stopped
```

Award 0 marks if the process is running in the background.

Code Quality [5]

27. Modularity and abstraction [3]
    Check for separate files, separate functions and separation of concerns. (A good place to check could be their parsing functions, different functions should ideally be responsible for different levels in the hierarchy - tokenization by space, by semicolons, by @ and $, by pipes)

28. Readme with a brief description of the system and what each file contains [1]

29. Makefile [1]

Viva [10]

30. Explain 2 code snippets [5]
    (Good places to ask include where they duplicate file descriptors for piping and redirection. They should be able to explain things like why do we create a backup of stdin and stdout before duplicating another file into them, and that why does the parent duplicate file descriptors before forking)

31. Explain their shell's output [5]
    We'll run these commands on their shell and ask them to explain why they behave the way they do. The outputs of these commands will vary based on their implementation.

    ```
    cd .. | ls # look at the pwd with the next prompt
    ```
    If they run each command of the pipeline in a separate sub-shell, the present working directory won't change, otherwise it will.

    OR (ask any one)

    ```
    setenv bored sleepy | env | grep bored # see if the variable was set or not
    ```
    Again, if the pipelined commands are run in a separate subshell, the child inherits a copy, and the parent won't see the change, otherwise it will.

32. Theory (Ask if the above commands fail to run on their shell or you have too much time :p)

Questions about file descriptors, processes, scheduling etc. Link to tut slides.

Bonus specification 1: Last working directory [5]

33. Correctness [5]

```
cd ..
cd -
cd /usr
cd -
```

Check for the directory changing and being printed correctly (whether it shows the absolute or the relative path or uses/does not use ~ does not matter).

Bonus specification 2: Exit codes [10]

34. Internal commands [1]

```
ls .. # successful
cd /usr # successful
ls does/not/exist # unsuccessful
pinfo 000 # unsuccessful
```

35. External foreground commands [2]

```
cat inp.txt # successful
cat doesnotexist.txt # unsuccessful
cat < inp.txt # successful
cat < doesnotexist.txt # unsuccessful
ps # successful
```

36. Send to the background [1]

```
sleep 100
CTRL+Z # unsuccessful
```

37. Interrupt (this depends on the process being interrupted) [1]

```
sleep 100
CTRL+C # it is unsuccessful for sleep
watch echo hi
CTRL+C # successful
```

38. The fg command [1]

```
sleep 5 &
fg 1 # should be successful after sleep ends

sleep 5 &
fg 1
CTRL+C # should be unsuccessful

fg 564 # should be unsuccessful
```

39. The bg command [1]

```
sleep 100
CTRL+Z
vim &
CTRL+Z
bg <sleep's job number> # successful
bg <vim's job number> # successful, even though vim doesn't
resume
bg 564 # unsuccessful
```

40. Run in the background [1]

```
sleep 10 & # successful
ps -Q & # successful, even though ps throws an error
bullshit & # successful
```

41. Semicolon-separated list [1]

```
ls; cd .. # successful
ls does/not/exist; cd .. # successful
ls; cd does/not/exist # unsuccessful
```

42. Pipeline [1]

```
cat num.txt | grep 321 # unsuccessful
cat num.txt | grep 12 # successful
rev < num.txt | rev # successful
rev < doesnotexist.txt | wc # successful
echo hi | rev < doesnotexist.txt # unsuccessful
```

Bonus specification 3: Command chaining [15]

43. Test cases

```
ls @ echo successful $ echo fail # should print successful and be a successful exit
ls does/not/exist @ echo successful $ echo fail # should print fail and still be a successful exit
ps -Q $ ps -Q $ ls @ echo penguins $ ls / # should execute the first 4 commands and be a successful exit
ps -Q @ ps -Q # should execute it once and be an unsuccessful exit
```

Try a few more combinations of commands listed in bonus specification 2 above.