

Assignment 1: Command-line

Complete assignment due: Thursday Sept. 4th, 9 p.m.

Problem 1 (50 points): Java command-line tool.

Write **onePipe.java**. The program will take pairs of command-line arguments. For each pair, the program will execute the first command, then take the standard output of the first command and provide it to the second command as standard input. This simulates a Unix pipe.

If the first command produces error messages on standard error, then do not start the second command. Instead, print the error messages.

For the second command, print the contents of standard error, if any, followed by the contents of standard out, if any.

For both commands, print the exit status.

Repeat for each pair of command-line arguments that are present.

Some examples:

```
$-> javac onePipe.java
$-> java onePipe "cat /home/cs352/fall08/books/gettysburg.txt" "wc"
Executing "cat /home/cs352/fall08/books/gettysburg.txt":
Nothing on standard error
Executing "wc":
Nothing on standard error
Standard output results for "wc":
    23    280   1495
Exit code for "cat /home/cs352/fall08/books/gettysburg.txt": 0
Exit code for "wc": 0
```

```
$-> java onePipe "ls -l" "cal 9 1752" "uptime" "wc -z"
Executing "ls -l":
Nothing on standard error
Executing "cal 9 1752":
Nothing on standard error
Standard output results for "cal 9 1752":
    September 1752
Su Mo Tu We Th Fr Sa
      1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

```
Exit code for "ls -l": 0
Exit code for "cal 9 1752": 0
```

```
-----

Executing "uptime":
Nothing on standard error
Executing "wc -z":
Standard error results for "wc -z":
wc: invalid option -- 'z'
Try `wc --help' for more information.
Nothing on standard output
Exit code for "uptime": 0
Exit code for "wc -z": 1

-----
```

Notes:

- For each command, print the contents of standard error, or state there is nothing on standard error.
- Print the contents of standard output for the 2nd command only (not the 1st), or state there is nothing on standard out.
- Print the exit status of each command.
- Following the results of each pair of commands, print a blank line, a line containing a row of 20 dashes, and another blank line.

Handling errors:

There are two types of errors that your program is expected to handle:

- If the number of command-line arguments is odd, then print an error message to standard error and exit with a status of **1**. For example:

```
$-> java onePipe "ls -la" "wc" "cal 1 1950"
Arguments must be paired
Usage: java onePipe cmd1 cmd2 [cmd1 cmd2] ...
$-> echo $?
1
```

- If the first command of a pair produces error output, the program will print the error output, the exit status of the first command, and then skip (not run) the second command. The program will continue with the next pair of command-line arguments. For example:

```
$-> java onePipe "ls -lz" "wc -lc" "cal 1 1950" "wc"
Executing "ls -lz":
Standard error results for "ls -lz":
ls: invalid option -- 'z'
Try `ls --help' for more information.
```

```
Exit code for "ls -lz": 2
```

```
-----
```

```
Executing "cal 1 1950":  
Nothing on standard error  
Executing "wc":  
Nothing on standard error  
Standard output results for "wc":  
      8      40     136  
Exit code for "cal 1 1950": 0  
Exit code for "wc": 0
```

```
-----
```

Java versions:

- On lectura, `/usr/bin/javac` and `/usr/bin/java` are version 1.6. If you use the command `javac` without the pathname, you should get this version by default.
- If you have a Mac running OS X, the current version of Java is 1.5, not 1.6. However, my solution to the problem works on both OS X and on lectura. (It is possible to get version 1.6 for OS X, but only if your Mac uses 64-bit, Intel CPUs.)

How to create sub-processes in Java:

The `java.lang.Runtime` class can be used to execute a sub-process. For example:

```
Process child;  
child = Runtime.getRuntime().exec("ls -la");
```

The `java.lang.Process` class has methods for getting the standard output and standard error streams from the sub-process. Note that the standard output of the `ls -la` sub-process above becomes an input stream to your Java program. Both are i/o streams, and it is usually best to treat them as buffered streams.

The standard input stream for a sub-process will be an output stream from your program. Data sent to this stream by your program will become the standard input for the sub-process. When there is no more data to send to the sub-process, use the `close()` method to send the necessary end-of-file to the sub-process.

The `waitFor()` method of the `Process` class has two uses:

- It will delay your program until the sub-process has terminated.
- It returns an integer which is the exit status of the sub-process. If the sub-process executed correctly, the exit status will be `0`. A non-zero exit status indicates that one or more problems occurred.

There is a script: `~cs352/fall08/assign1/testAssign1.ksh`. You can copy this script to your directory and use it to test your program. We will use these test cases in testing your program. We reserve the right to use additional test cases. Here is an example:

```
$-> cp ~cs352/fall08/assign1/testAssign1.ksh .
$-> testAssign1.ksh
Compiling onePipe.java
running test #1:
java onePipe "cat /home/cs352/fall08/books/gettysburg.txt" "wc"
passed.

running test #2:
java onePipe "cal 9 2008" "tr 0123 abcd"
passed.

running test #3:
java onePipe "cat /home/cs352/fall08/books/gettysburg.txt" "wc" "cal
9 2008" "tr 0123 abcd"
passed.

running test #4:
java onePipe
passed.

running test #5:
java onePipe "cal 9 2008"
passed.

running test #6:
java onePipe "ls -la" "cal 9 1752"
passed.

running test #7:
java onePipe "fgrep Patrick /home/cs352/fall08/books/sawyer.txt"
"wc"
passed.

running test #8:
java onePipe "ls -la" "cal 9 1752" "fgrep Patrick /home/cs352/
fall08/books/sawyer.txt" "wc"
passed.

running test #9:
java onePipe "ls -la" "fgrep Patrick /home/cs352/fall08/books/
sawyer.txt"
passed.
```

```

=====
=====
tests passed: 1 2 3 4 5 6 7 8 9
tests failed:
=====
=====

```

If your program fails to pass one or more of the test cases, the output will show the difference(s) between your output and the output of our solution.

The various `~cs352/fall08/assign1/patrick*` files contain the results of running my solution on each testcase.

Turnin: Use the `turnin` program to turn in your `onePipe.java` program. The command is:
`turnin 352assign1 onePipe.java`

See the man page for the `turnin` program for details on what `turnin` can do and how you can confirm that your file was turned in.

Problem 2 (25 points): vi.

Copy the file `/home/cs352/fall08/assign1/problem2.txt` to your directory. Answer the `vi` questions that are contained in the file.

It is not required that you actually use `vi` to answer the questions. It is recommended, so that you can gain some experience with `vi`. If you choose not to use `vi`, and you use a Windows computer, then there is an extra step needed. After you have edited the file on your Windows system, upload it to your CS account. Then, on lectura, use the `dos2unix` program on the file before you turn it in. There is a manpage for `dos2unix`. (If you use `vi` on lectura to complete this problem, you do not need to use the `dos2unix` program.)

Turnin: Use the `turnin` program to turn in your `problem2.txt` file. The command is:
`turnin 352assign1 problem2.txt`

See the man page for the `turnin` program for details on what `turnin` can do and how you can confirm that your file was turned in.

Problem 3 (25 points): Simple script.

You will write a Korn shell script named `problem3.ksh`. The script will consist of ten lines. The first line will be:

```
#!/bin/ksh
```

The second through tenth lines will contain your answers to parts a through e below. Each question is to be done with exactly one Unix command-line. In-between the answers put the statement:

```
echo
```

This will put a blank line between each answer. Note that you will end up with 5 commands for parts a.) through e.) and 4 **echo** statements.

a.) Use **ls** to show the long listing format for all entries in the directory **~cs352/fall108/Cexamples**, including the “hidden” files. Arrange the files in order by time of last access with the oldest access time first.

b.) Use the long listing form of **ls** and **fgrep** to list all the entries in the directory **/cs/www/classes/cs352/fall108** that contain 3 consecutive blank spaces. Filter out those lines that have 4 (or more) consecutive blank spaces.

c.) Use **cat** and **wc** to count the total number of lines in the files in **~cs352/fall108/assign1** whose names start with **patrick**. Note your answer is to print only the total number of lines, not the number of lines in each individual file.

d.) Use **cat** to create a file in the local directory named **twoFiles.txt** that contains the contents of the files **gettysburg.txt** and **raven.txt** (in that order). **gettysburg.txt** and **raven.txt** can both be found in **~cs352/fall108/books**. Append to **twoFiles.txt** the output of **wc** on the file **twoFiles.txt**. Print the last 20 lines of the file **twoFiles.txt**.

e.) Re-arrange the lines from the help option of **wc** so that all the lines that contain **--** will be printed first, followed by a blank line, followed by the remaining lines (those that do not contain **--**).

It is not required that you use **vi** to write your script. It is recommended, so that you can gain some experience with **vi**. If you choose not to use **vi**, and you use a Windows computer, then there is an extra step needed. After you have edited the file on your Windows system, upload it to your CS account. Then, on lectura, use the **dos2unix** program on the file before you turn it in. There is a manpage for **dos2unix**. (If you use **vi** on lectura to complete this problem, you do not need to use the **dos2unix** program.)

Turnin: Use the **turnin** program to turn in your **problem3.ksh** script. The command is:
turnin 352assign1 problem3.ksh

See the man page for the **turnin** program for details on what turnin can do and how you can confirm that your file was turned in.

Final Summary:

There are three files that you will need to turnin for full credit on this assignment:

onePipe.java
problem2.txt
problem3.ksh