CSc 352 (Fall 2018): Assignment 11

Due Date: 11:59PM Wed, Dec 4

The purpose of this assignment is to get some experience write bash shell scripts.

General Requirements

1. Your programs should indicate if they executed without any problems via their *exit status*, i.e., the value returned by the program when it terminates:

Execution	Exit Status
Normal, no problems	**0**
Error or problem encountered	**1**

2. Under *bash* you can check the exit status of a command or program ***cmd*** by typing the command "**echo $?**" immediately after the execution of ***cmd***. A program can exit with status *n* by executing "**`exit(`***n***`)`**" anywhere in the program, or by having **`main()`** execute the statement "**`return(`***n***`)`**".
3. Remember your code will be graded on lectura using a grading script. You should test your code on lectura.

Testing

Unlike previous assignments, this program will be a bash shell script. Since bash scripts are interpreted I cannot provide you with an example executable as I have when you were writing C programs. Also, it is more important than ever that you test your code on lectura since different implementations of bash might not work exactly the same.

I have placed, in /home/cs352/fall18/assignments/assg11/prob1 an example output file to use for testing. In file `exampleOut` is what your output should look like when you test the executable `anagrams2` using the test directory `testcases`. Both the executable and test directory are also provided in the same location.

Submission Instructions

Your solutions are to be turned in on the host **lectura.cs.arizona.edu**. Since the assignment will be graded by a script, it is important you have the directory structure and the names of the files exact. Remember that UNIX is case sensitive, so make sure the capitalization is also correct. For all our assignments the directory structure should be as follows: The root directory will be named **assg#**, where # is the number of the current assignment. Inside that directory should be a subdirectory for each problem. These directories will be named **prob#** where # is the number of the problem within the assignment. Inside these directories should be any files required by the problem descriptions.

For this problem the **prob1** directory will contain a bash script named **testCode**. Be sure to set it's executable bit on and that its first line tells the shell it should be run as a bash script.

To submit your solutions, go to the directory containing your assg11 directory and use the following command:

turnin cs352f18-assg11 **assg11**

Problems

prob1: testCode

In this problem you will write a bash script for testing C programs.

You will write a bash script "testCode". This script will test a specified program using a reference executable and test files found in a specified directory. One way to use the script would be to specify those two arguments on the command line as:

`testCode` ***`program_name`*** ***`test_case_dir`***

For example

`testCode anagrams2 testcases`

The relevant files in *test_case_dir* will be named test*_program_name**, where *, like the bash wildcard, is any string, including nothing. It should be OK if there are other files in that directory (e.g., test cases for a different program). These should simply not be used. Finally, the reference executable must be called ex*_program_name* and should also exist in the directory *test_case_dir*. In our example the test cases would be named things like `test_anagrams2_01`, `test_anagrams2_02`, `test_anagrams2bigTest`, and the reference executable would be named `ex_anagrams2`. All these files would live in the directory `testcases`. Note, that we specified a relative path to a directory here, but an absolute path should also work. The program being tested (*program_name*, or in or example, `anagrams2`) should be located in the current directory, not the directory with the test cases.

Command line arguments are convenient in many cases, but we might want to test the script itself using input from standard input. Hence you also need to implement the following. If the command is invoked with no arguments, you must read the arguments from **stdin** one line at a time. They should be assumed to be in the same order as the arguments are listed. (i.e. The program name is read first and then the test case directory.) If you do not get enough arguments after all that, print an error message and exit with a status of 1. You do not need to worry about the possibility of too many arguments.

Your script should run the reference executable as well as the specified executable for each testcase. These programs should be run using redirection to have them use the testcase as their stdin. You do not need to include any command line arguments for these programs. You need to run the executable (the one in the current directory, not the reference executable) in such a way that if there is an infinite loop it will time out (see below). You need to report "time out" termination as well as abnormal execution (see below), or, if things go well enough so that those issues are not relevant, then you want to analyze stdout, stderr, and return codes. Finally, you will run valgrind and check for both memory errors and memory leaks (see below).

- **Error output**

  Your script should report issues regarding not being able to continue (e.g., it cannot find the reference executable) to standard error, and exit with non-zero status. These errors should not be confused with errors in the program (e.g., not matching the reference executable output). Errors in the program being tested go into the report to standard output described next. If your script successively tests the program, even if it finds that there are problems with it, then you should return zero.

- **Standard output**

  For each test case you will print one or more result lines to stdout. The grading script will ignore blank lines or changes in white space, but make sure the text of the outputs are correct. If the program times out (see below) on a test case, then all that is printed is:

  ```
  *** testcase: test_xxx_01 [ FAILED – Timed out ]
  ```

  Note that test_xxx_01 is the name of the file without any "/". When a program crashes for some reason (see below) on a test case all that is printed is:

  ```
  *** testcase: test_xxx_01 [ FAILED – Abnormal termination ]
  ```

  If a program terminates normally, then print how it did for each thing being tested. For example:

  ```
  *** testcase: test_xxx_01 [ stdout - PASSED ]

  *** testcase: test_xxx_01 [ return code - PASSED ]

  *** testcase: test_xxx_01 [ stderr - FAILED ]

  *** testcase: test_xxx_01 [ valgrind - FAILED ]

  *** testcase: test_xxx_01 [ memory free - PASSED ]
  ```

  Where you print "PASSED" or "FAILED" based on whether the program passes or fails that part of the test. Here are the criteria for each part:

  stdout – passes if the stdout's match (use –Z flag for diff)

  return code – passes if the return codes match

  stderr – passes if either both the reference executable and the program tested write something to stderr or they both write nothing to stderr

  valgrind – passes if valgrind says there are no memory errors

memory free – passes if valgrind says all the memory has been freed upon program exit.

- **A few hints for some of the trickier parts**

  You probably want to use the program "timeout" to run your program (see man "timeout"). We will assume a timeout of 2 seconds for this assignment. If timeout successfully runs the program, it will report the return code of the program that it run, so you can use that for the return code checks. But, if the program that timeout is running for you times out, timeout will exit with 124. Finally, if the program exits abnormally due to a system signal (we have not studied this yet), it will return 128+NUM, where NUM is the signal number. You do not need to fully understand this convention. You can just assume that a return greater than 128 means your program died for some reason. But you do need to check for it because your program might die, for example, due to a segfault. (The signal there is SIGSEGV which is 11, so the return would be 139). Such return codes from timeout covers the "Abnormal termination" case.

  For this script, you should use grep to pull what you need from the output of valgrind. You may want to look at the man page for grep or look it up online. You don't need to know much about grep to have enough knowledge for this assignment.

  When using timeout if your program crashes, timeout will send an error message that you'll want to ignore. Unfortunately you're probably already redirecting the output from your program, so how do you say to not print these messages? It turns out this works:

```
{
  timeout <rest of your command here>
} &> /dev/null
```

  The <rest of your command> part might (should) include redirections for the output from the code you're running. The { } tell bash to just redirect the output from timeout to /dev/null.

- **Script failures**

  We expect error messages on standard error, and non-zero termination, for:

  - Failure to read (either as parameters or from stdin) a program name and a test case directory.
  - The test directory doesn't exist.
  - Cannot execute the reference executable
  - Cannot execute the program to be tested

- **More hints**
  - There is an exit command not mentioned in the slides which you might find useful. (see man page)

- You will need to use redirections to create files to do some of these tasks. Assume your script can write to the directories. Choose files names that you might expect not to collide with existing files. There is no way to guarantee this and we won't worry too much about this. Just don't redirect to a file named after one of our program names (e.g. don't redirect to anagrams2 or something like that). You will want to look at the man page for test to see how for empty files.
- Finally, remember if you want to suppress the output of a command you can redirect to /dev/null