Lab 5: `bash 2`

Just have fun with it!

Please Read the entire assignment before you start coding.

This is an in-class lab. **You can work on this in groups up to 3**, but only one group member should submit the script into Canvas and **mention in the script comments and Canvas with whom you worked.**

You will write a `bash` shell script to control and report on the behavior of another program. **A method like this is often used when testing the output from other programs**. In this case, the "other program" is the program you want to test. It is called `hydra`.

For our purposes, you've been given a program called `hydra` (**it's a compiled binary file only, no source code**). You really don't know or care how **`hydra`** is implemented, you only know how to pass an option on the command line and **how to interpret the documented exit status of the program**. The program **`hydra`** takes **a single numeric argument on the command line**. For example, you could run the program as follows:

```
./hydra 41
```

The **exit status** (sometimes called the return value from the program) of the program can be interpreted as follows (Table 1):

Exit status of `hydra`	Meaning
0	The program was successful
1	You did not supply a command line argument
2	You supplied an invalid command line argument
3	The program failed for some other reason

Table 1: Documented exit values and meaning for `hydra`.

Do you remember the special shell variable that contains the exit status of the program? This would be an especially good time to review the slides about `bash` shell scripting. If you really studied those slides, this assignment will be easy.

Try the following and view the exit value of hydra each:

```
./hydra 7
./hydra
./hydra a
./hydra 8
```

The program `hydra` is found in

```
~rchaney/Classes/cs333/Labs/Lab5
```

Copy the `hydra` file into your directory:

```
cp ~rchaney/Classes/cs333/Labs/Lab5/hydra .
```

Yes, that is a dot at the end of the command. **It is required**.

Your `bash` shell script should have the *effect* of the following (written in something like a *pseudo*-C language).

```
for ( num = begin; num <= end; num++ ) {
    if ( hydra(num) == 0 ) { // get the exit value of hydra
         printf("%d was successful\n", num);
    }
}
```

Your shell script (which must be called `run_test.bash`) must use either the `getopt` command or the `getopts` bash built-in for parsing the command line (I highly recommend the `bash` built-in `getopts`). **Your script must accept all 4 of the command line options shown in Table 2**.

Command line Option	Function
-h	Output a help message and exit with value `0`.
-e <end value>	Specified the **end** value for the test.
-b <begin value>	Specified the **begin** value for the test.
-v	Run in verbose mode – **additional info should be printed for commands that fail**. See (on page 4) for an example of this output. Did you notice how with the `-v` option, each value run with the `hydra` program produces output?

Table 2: Command line options for the `run_test.bash` shell script.

Requirements for handling the options from the command line

1. If neither the `-b` nor the `-e` options are specified on the command line, your script should behave as if the `-h` option was specified.
2. If the `-e` option is not specified, the default value of 10 should be used for the end value and continue. If the value given with `-e` is less than 1, then set the end value to 10 and continue. If the `-e` occurs more than once on the command line, the script should use whatever value was **last given** (this will happen automatically if you use `getopts`, so you really don't have to worry about how to implement it).
3. If the `-b` option is not specified, the default begin value of 1 (one) should be used and continue. If the value given with `-b` is less than 1 (one), set the begin value to 1 (one) and continue. If `-b` occurs more than once on the command line, the script should use whatever value was **last give**n.

4. You must test your script with values from 1 up to 100.

5. If the value specified with `-b` is greater than the value given with `-e`, call the `show_help` function and exit.
6. When the `-h` command line is given, your script should **call a function** called `show_help` that displays the following message **and then exits with value 0** (with the date shown as today's date):

```
./run_test.bash [-h] [-b <begin>] [-e <end>] [-v]
  Today's date is  Mon Nov 27 03:31:18 AM PST 2023
```

You must be able to give the command line options in any order. The `getopts` function will do this for you (as will `getopt`). Review , page 4 below, to see several examples of running the `bash` script. The `script_tests.txt` also has many examples of running the script.

Let's Write a Script!

Don't get intimidated by this assignment. Break it down it down into small parts and complete each part. For example,

1. Look at the tables and figures in the assignment.
2. **Read item 18 below. Check this frequently.**
3. You may find the slides on writing `bash` scripts really handy.
4. You may find the examples `bash` scripts mentioned in the slides helpful (directory ~rchaney/Classes/cs333/src/bash), such as `hello_world_GETOPTS.bash`.
5. Create a directory where you will work on the assignment.
6. Copy the `hydra` executable and the file `script_tests.txt` into that directory. The `script_tests.txt` file gives you several examples of how your script can be run.
7. With your editor of choice, begin writing your `run_tests.bash`. Make sure you set the execute bits enabled on your shell script. Remember to set the execute bits on your shell script before you try to run it (the `chmod` command). **And, do NOT be lazy and just 777 it**.
8. I recommend that you create variables and initialize them with a value that makes it easy to determine of the value has been changed when processing the command line. I had a fondness for the value zero. Read Requirement #1. Good names for the variables might be: `BEGIN`, `END`, and `VERBOSE`.
9. Focus first on handling the command line options (using `getopt` or `getopts` (BTW, I **STRONGLY** recommend `getopts` over `getopt` for this)) and setting up some variables to hold the various values. You probably want variables to hold the `BEGIN` value (the `-b` option), the `END` value (the `-e` option), and some indicator about `VERBOSE` (the `-v` option). You probably want to deal with the `-h` option at the end of this section in your code.
10. Until you completely process the command line, you can't try to run `hydra`.
11. Write a few simple if-then statements to satisfy the requirements for handling the variables after the `getopts` loop. Look back at the requirements, just below Table 2, and work down through those. Don't try every possible permutation of variables. Focus on what is stated in the requirements.
12. Now just write a loop that runs the test program (`hydra`), captures and tests its exit status, and displays a message about success or other. I used a `for` loop that looks a lot like a C `for` loop. Look at `assocArray2.bash` script for some inspiration. Writing this out as pseudocode might really help you through this part.
13. **Be sure to check to see if you have a bunch of unwanted processes running. The `ps` command is great for this. Try something like:**

```
ps -ef | grep ${LOGNAME}
```

14. **Test your script with values from 1 through 100.**
15. Don't try to be toooooo fancy. Just get it done. There may be more than one way to accomplish portions of this script. Choose any `bash` script method that works.
16. You will have about 70-75 or lines of `bash` code when you are done. Submit the script into Canvas to the `Lab5` assignment (**it does not need to be a gzipped tar file, just a `bash` script**).
17. **You can work on this in groups of up to 4**, but only one group member should submit the script into Canvas and **mention in the script comments with whom they worked**.

18. **Be sure to check to see if you have a bunch of unwanted processes running. The `ps` command is great for this. Try something like:**

```
ps -ef | grep ${LOGNAME}
```

```
 babbage ~/Classes/cs333/Labs/Lab5
rchaney # ./run_test.bash -h
./run_test.bash [-h] [-b <begin>] [-e <end>] [-v]
  Today's date is  Thu May 22 01:09:52 PM PDT 2025
 babbage ~/Classes/cs333/Labs/Lab5
rchaney # ./run_test.bash -b4
    5 was successful
    7 was successful
 babbage ~/Classes/cs333/Labs/Lab5
rchaney # ./run_test.bash -e4
    2 was successful
    3 was successful
 babbage ~/Classes/cs333/Labs/Lab5
rchaney # ./run_test.bash -b3 -e12
    3 was successful
    5 was successful
    7 was successful
    11 was successful
 babbage ~/Classes/cs333/Labs/Lab5
rchaney # ./run_test.bash -e4 -v
Begin value = 1    End value = 4
    1 returned 2
    2 was successful
    3 was successful
    4 returned 3
 babbage ~/Classes/cs333/Labs/Lab5
rchaney # ./run_test.bash -v -b 3 -e 5
Begin value = 3    End value = 5
    3 was successful
    4 returned 3
    5 was successful
 babbage ~/Classes/cs333/Labs/Lab5
rchaney # ./run_test.bash -v -b 3 -e 5 -b 4 -e 7
Begin value = 4    End value = 7
    4 returned 3
    5 was successful
    6 returned 3
```

Final note

The labs in this course are intended to give you basic skills. In later labs, we will assume that you have mastered the skills introduced in earlier labs. **If you don't understand, ask questions.**

Last note: Sometimes people don't follow a perfectly good specification.