CS 279 - Homework 3

Deadline

11:59 pm on **Friday, September 23**

Purpose

To practice with the `bash` shell's versions of local variables, shell interactive input, `if` statements, `for` loops, command-line arguments, and more.

How to submit

You will complete **Problems 1 and 2** on the course Canvas site (short answer questions on local shell variable basics and on built-in variables for command-line arguments), so that you can see if you are on the right track.

For the rest of the problems, you will create several files and then submit those to the course Canvas site.

NOTE: While I list the separate files you need to submit for each problem below, I am going to set up Canvas to *also* accept `.zip` files.

That is,

- you can submit each file to Canvas,
- OR, if you prefer, you may compress your files to be submitted into a single `.zip` file and submit that `.zip` file to Canvas.

Important notes

Assume, for all `bash` scripts in this course, that the following are required:

- Start each script with the line that is considered good style (and is a CS 279 course requirement), that specifies that this script should be executed using the `bash` shell
- After a blank line, put in one or more **comments** including at least the name of the shell script, your name, and its last modified date
- And follow these comments with a blank line.

Problem 1 - 14 points

Problem 1 is correctly answering the "HW 3 - Problem 1 - Short-answer questions on local shell variable basics" on the course Canvas site.

Problem 2 - 12 points

Problem 1 is correctly answering the "HW 3 - Problem 2 - Short-answer questions on built-in variables for command-line arguments" on the course Canvas site.

Problem 3

FUN FACTS:

- `[ -d` *filename* `]` # will be true if *filename* is a directory file
- `[ -f` *filename* `]` # will be true if *filename* is a "regular" file (not a directory or a device file)

We're going to assume, for our purposes here, that someone using this problem's script will NOT have any device files in the directories they are using this script in!

Write a bash shell script named `file-type` or `file-type.sh` that meets the following requirements:

- This script expects exactly one command-line argument, no more and no less. If it is not given exactly one command-line argument, it should complain and exit, with an exit status of 1.
- Otherwise:
  - if the single command-line argument is the name of a regular file, this should print a message to the screen saying this is a regular file (including the file's name)
  - otherwise, if the single command-line argument is the name of a directory, this should print a message to the screen saying this is a directory (including the directory's name)
  - otherwise, we'll assume that means this is not a file in the current directory, and this should print a message to the screen saying this (including the non-existent name).

Then, demonstrate its use as follows:

- Call your script with no arguments, redirecting the result to `p3-no-args.txt`
- Call your script with two or more arguments, redirecting the result to `p3-too-many-args.txt`
- Call your script with the name of a non-existent file, redirecting the result to `p3-not-exists.txt`
- Call your script with the name of a directory file, redirecting the result to `p3-dir-file.txt`
- Call your script with the name of a regular file, redirecting the result to `p3-reg-file.txt`

Submit your resulting files:

- `file-type` or `file-type.sh`
- `p3-no-args.txt`
- `p3-too-many-args.txt`
- `p3-not-exists.txt`
- `p3-dir-file.txt`
- `p3-reg-file.txt`

Problem 4

Write a bash shell script named `gen-fodder` or `gen-fodder.sh` that meets the following requirements:

- It should use a "classic"-style `for`-loop to create in the current directory the 20 files `test1.txt`, `test2.txt`, ... `test20.txt`,
  - each containing one line containing the file's name,
  - and one line containing any line of text you'd like that also includes the file's number (`test1.txt` includes a `1` in its second line, `test2.txt` includes a 2 in its second line, etc.)

Then, demonstrate its use as follows:

- create an empty new directory
- in that directory, run:
  - echo an "about to start test" message into a file `p4-gen-test.txt`

  - append the results of an `ls` command to `p4-gen-test.txt` to show the directory's initial contents
- now run `gen-fodder.sh` in that directory
- and afterwards, echo an "after test" message and append it to `p4-gen-test.txt`
- and then append the results of another `ls` command to `p4-gen-test.txt` to show the directory's resulting contents (should be considerably more files now)

Submit your resulting files:

- `gen-fodder` or `gen-fodder.sh`
- `p4-gen-test.txt`

Problem 5

Write a bash shell script `use-fodder` or `use-fodder.sh` that meets the following requirements:

- It should use a "list-style" `for`-loop that uses a back-quoted command with file globbing using the wildcard character * to list ONLY the file names that start with `test` and end with `.txt`,
  - for each of these files, append the line "`TAG gotcha`" to the end of these files

To demonstrate this script, do the following:

- go to the directory you used to test `gen-fodder.sh` in Problem 4 -- it should still contain the files `test1.txt`, `test2.txt`, ... `test20.txt`
- echo "`directory contents:`" into a file `p5-use-test.txt`
- list the files currently in this directory, appending the result to `p5-use-test.txt`
- run `use-fodder.sh` in this directory
- echo "`after ran use-fodder.sh:`", appending the result to `p5-use-test.txt`
- then, run this command (which we will be discussing in more detail later!):

```
grep "TAG gotcha" * >> p5-use-test.txt
```

Submit your resulting files:

- `use-fodder` or `use-fodder.sh`
- `p5-use-test.txt`

Problem 6

FUN FACTS:

- `[ -e` *filename* `]` # will be true if *filename* is the name of a file that currently exists
- `[ -d` *filename* `]` # will be true if *filename* is a directory file
- `[ -f` *filename* `]` # will be true if *filename* is a "regular" file (not a directory or a device file)
- You can precede any of the above with `!` to negate them -- that is,

  `[ ! -e` *filename* `]` # will be true if *filename* is NOT the name of a file that currently exists

For Homework 2 - Problem 5, you wrote a script `backup-all` or `backup-all.sh` that copied all non-directory files to a local directory named `BACKUP`.

But -- with interactive input and command-line arguments now in your `bash` toolbox, you can make a more versatile and less "clunky" version of this script, that can avoid certain errors more gracefully.

Write a `bash` shell script named `backup-to` or `backup-to.sh` that meets the following requirements:

- Find out where the non-directory regular files in the current working directory should be backed up to:
  - if NO command-line arguments were given, ask the user to enter the name of the directory to which to back up the files, and read in what they enter
  - otherwise, assume the first command-line argument is the name of the directory to which to back up the files.
- If a file with the name requested for the back-up directory currently exists and is not a directory file, complain and exit with an error status of 1.
- If a file with the name requested for the backup directory does NOT currently exist, create it.
- Now try to copy all non-directory files to this desired directory to back up to -- BUT use a loop and an `if` statement so that you ONLY attempt to copy *regular* files to this directory to back up to!
  - (so, you should not get the errors messages from attempting to copy directories over, for example)
- Finally, it should echo to the screen a descriptive message indicating that it is about to show the current contents of the requested backup directory,
  - ...and then it should output to the screen the current contents of that requested backup directory.

Also perform at least the following tests of `backup-to`/`backup-to.sh` in a directory containing at least 3 non-directory files:

- Call it with an existing non-directory file as its command-line argument, to show that it complains and exits as expected, redirecting the script's output into a file `p6-non-dir-test.txt`
- Call it with a currently-not-existing name as its command-line argument, to show it creates that as a backup directory and copies all the non-directory local files into it, redirecting the script's output into a file `p6-new-dir-test.txt`
- Create a new text file `new-file.txt`, and then call your script with the same directory name as used in the previous test (to show this also works), redirecting the script's output into a file `p6-existing-dir-test.txt`
  - Note that you should see `new-file.txt` in its contents in `p6-existing-dir-test.txt`

(Because any prompts for interactive input will also be redirected when you redirect a script's output to a file, I cannot think of a reasonable way for you to demo that you've tested your script's behavior when no command-line argument is given! But you should test your script and make sure it also behaves for those cases, also.)

Submit your resulting files:

- `backup-to` or `backup-to.sh`
- `p6-non-dir-test.txt`
- `p6-new-dir-test.txt`
- `p6-existing-dir-test.txt`