CS 279 - Homework 9

Deadline

11:59 pm on **Friday, December 2**

Purpose

To answer questions about more history-searching options and file links, and to practice a bit more with file links and Bash functions.

How to submit

You will complete **Problems 1 and 2** on the course Canvas site.

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 *(EXCEPT for a script containing JUST Bash functions)* 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 - 8 points

Problem 1 is correctly answering the "HW 9 - Problem 1 - Short-answer questions involving other history-searching options" on the course Canvas site.

Problem 2 - 8 points

Problem 1 is correctly answering the "HW 9 - Problem 2 - Short-answer questions involving links" on the course Canvas site.

Problem 3

In a file `hw9-3.txt`, include:

- your name
- the part you are giving an answer for
- the answers specified below

3 part a

On nrs-projects, within a directory other than your home directory:

- Create a *regular* file **`some-commands.txt`** whose contents include the names of at least three UNIX/Linux commands.
- Now create a *symbolic/soft* link named **`ref-s-commands.txt`** that refers to your file `some-commands.txt`.
  - As long as it is not within your home directory, this symbolic link can be in the same directory or in a different directory, your choice!
- Now create a *hard* link named **`ref-h-commands.txt`** that refers to your file `some-commands.txt`.
  - As long as it is not within your home directory, this hard link can be in the same directory or in a different directory, your choice!
- Write one or more **`ls -li`** commands to show the names and i-node numbers for `some-commands.txt`, `ref-s-commands.txt`, and `ref-h-commands.txt`.

For your answer for this part, paste the results of this/these **`ls -li`** commands.

3 part b

Fun fact: We have seen that the `find` command has an option `-type`, and when used with `d` this can be used to find directories. So, to find and print the names of all directories reachable from the current directory, this works:

`find . -type d -print`

It turns out that you can use `-type` with `l` (that's an el, not a one) to find *symbolic* links!

SO: for your answer for this part, write a `find` command that will find and print the names of all symbolic links reachable from your nrs-projects home directory (even if you run this command in a directory other than your home directory).

3 part c

Run your command from 3 part b on nrs-projects -- it should find at least your symbolic link that you created in 3 part a (and possibly more, depending for example on whether you were the "driver" for the Week 13 Lab Exercise!)

Note that it should **not** include your `ref-h-commands.txt` in its output -- this command will not find hard links.

For your answer to this part, paste in the result of running `ls -l` with the full pathname your `find` command returns for `ref-s-commands.txt`.

3 part d

Call `wc` for `ref-h-commands.txt`, `ref-s-commands.txt`, and `some-commands.txt`, and paste in the results as the first part of your answer for this part.

Using `nano`, add the name of at least one additional UNIX/Linux command of your choice to `some-commands.txt` and save that change.

Now, again call `wc` for both `ref-commands.txt` and `some-commands.txt`, and paste in the results as the second part of your answer for this part.

3 part e

So -- are symbolic links really used very much in a UNIX/Linux system?

You can find out how many you can reach from nrs-projects' root -- try the following command on nrs-projects:

```
find / -type l -print 2> /dev/null | wc -l
```

Paste its output (the number of lines in its result) as the first part of your answer to this part.

One of these symbolic links is `/usr/java/latest`.

Paste the result of calling:

```
ls -l /usr/java/latest
```

...as the last part of your answer to this part.

Submit your resulting files `some-commands.txt` and `hw9-3.txt` .

Problem 4

Create a shell script `hw9-functions.sh` that contains the following two Bash functions.

- Because it will be included in other shell scripts using `source`, do **not** start it with the usual `#!/bin/bash` -- but do still include the usual shell-script comment(s)!

4 part a

Hm! It turns out, for all the things that are easy to test in a Bash shell script, testing whether something is an integer takes a bit of a kluge!

It looks like one approach is to use a regular expression to do this. For our purposes, we in particular would like a non-negative integer, and we'll assume that it is **not** to be preceded by a +.

Write a Bash function `is_quant` that meets the following specifications:

- We'll be discussing the rather odd meaning of `return` in Bash functions -- and for this function, have it `return` the desired exit status. (That is, `return 0` or `return 1` rather than `exit 0` or `exit 1`)
- We'd like it to "silently" work -- it won't output anything to standard output, it will simply `return` with an appropriate exit status.
- if exactly one command-line argument is not given, it returns a non-zero exit status of your choice
- otherwise, it should use a regular expression in an `if` statement to return an exit status of 0 (success!) if the input is indeed an (unsigned) integer greater than or equal to 0, and to return a non-zero exit status of your choice if the input is not.

Note that you can test this by typing something like:

```
source hw9-functions.sh
is_quant 47
echo $?
```

...to see what exit status it returned.

4 part b

Remember the overly-trusting function `make_line` from the Week 13 Lab Exercise?

- "expects a string to repeat and a number of repetitions, and echoes to standard output a single line

containing that string repeated that many times."

- For example,

  `make_line Moo 4`

  ...would cause the following to be echoed to the screen:

  `MooMooMooMoo`

Modify your function `make_line` from the Week 13 Lab Exercise as follows:

- This version should check the number of arguments it is called with, and if it is not called with exactly 2 command-line arguments, it should `return` a non-zero exit status of your choice.
- This version should also use 4 part a's function `is_quant` to help it to verify that the second argument is a non-negative integer, and it should `return` a non-zero exit status of your choice if it is not.
- Otherwise, it indeed echoes to standard output the desired single line to standard output containing that string repeated that many times.

Note that you can test this by typing something like:

```
source hw9-functions.sh
make_line Moo 4
echo $?
make_line Moo Baa
echo $?
```

...to see its output (if any) and what exit status it returned.

Submit your resulting file `hw9-functions.sh`.

Problem 5

Create a shell script `hw9-test.sh` that tests your functions from Problem 4, meeting the following requirements:

- It should use the `source` command with `hw9-functions.sh` to make Problem 4's functions available to this shell script.
- Test function `is_quant`:
  - Echo that you are about to call `is_quant` with no arguments, do so, and then echo that call's returned exit status to the screen in a message of your choice.
  - Echo that you are about to call `is_quant` with more than one argument, do so, and then echo that call's returned exit status to the screen in a message of your choice.
  - Echo that you are about to call `is_quant` with an unsigned integer, do so, and then echo that call's returned exit status to the screen in a message of your choice.
  - Echo that you are about to call `is_quant` with an argument that ISN'T an unsigned integer, do so, and then echo that call's returned exit status to the screen in a message of your choice.
  - (if you would like to add additional tests of your `is_quant` along with the above, that's fine and encouraged!)
- Test your now-less-trusting function `make_line`:

- Echo that you are about to call `make_line` with the wrong number of arguments, do so, and then echo that call's returned exit status to the screen in a message of your choice.
- Echo that you are about to call `make_line` with two arguments, but with a NON-number as the second argument, do so, and then echo that call's returned exit status to the screen in a message of your choice.
- Echo that you are about to call `make_line` with two GOOD arguments, and then do so -- BUT for this call, set a VARIABLE to the **backquoted** result of calling this function with those good arguments, and then echo the value of that variable afterwards to the screen in a message of your choice.
- (if you would like to add additional tests of your `make_line` along with the above, that's fine and encouraged!)

Submit your resulting file `hw9-test.sh`.

Submit your resulting files:

- `some-commands.txt`
- `hw9-3.txt`
- `hw9-functions.sh`
- `hw9-test.sh`