School of Computer Science, McGill University

COMP-206 Introduction to Software Systems, Winter 2022

Mini Assignment 2: Bash scripting

Due Date Feb 11, 18:00

This is an individual assignment. You need to solve these questions on your own.

You MUST use `mimi.cs.mcgill.ca` to create the solution to this assignment. An important objective of the course is to make students practice working completely on a remote system. Therefore, you must not use your Mac command-line, Windows command-line, nor a Linux distro installed locally on your laptop. You can access `mimi.cs.mcgill.ca` from your personal computer using **ssh** or **putty** as seen in class and in Lab A. **If we find evidence that you have been instead using your laptop, etc., to do some parts of your assignment work, you might loose all of the assignment points.** All of your solutions should be composed of commands that are executable in `mimi.cs.mcgill.ca`.

For this assignment, you will have to turn in two shell scripts. Instructors/TAs upon their discretion may ask you to demonstrate/explain your solution. No points are awarded for commands that do not execute at all. (Commands that execute, but provide incorrect behavior/output will be given partial marks.) All questions are graded proportionally. This means that if 40% of the question is correct, you will receive 40% of the grade.**TAs WILL NOT modify your scripts in anyways to make it work.**

Please read through the entire assignment before you start working on it. You can loose up to 3 points for not following the instructions in addition to the points lost per questions.

Lab C provides some background help for this mini assignment.

Total Points: 20

Ex. 1 — A custom archive script(9 Points)

1. Make sure that your script starts with a sha-bang to execute it using **bash** and is followed by a small comment section that includes your name, department and email id (format of this comment section is up to you). The script should also have additional comments for the important part of the code. **-1 point** if not followed.
2. **(5 points)** Your first task is to create a script which we will call `coderback.bash`. This script will take an individual file or directory, and back it up into a tar file to save your assignment and lab works frequently. Specifically, this script will take two inputs:
   1. the directory where the tar file should be saved;
   2. an individual file or directory to backup.

   Furthermore, the name of the tar file created will need to contain the name of the directory or file and the date the backup was created in `YYYYMMDD` format. Finally, the script will need to terminate with **error code 0** upon success and the appropriate error code otherwise (see below).
   For example, let's imagine that the following is executed on January 30, 2022:

   ```
   $ ./coderback.bash $HOME/mystash asgn1
   ```

   where `mystash` and `asgn1` are directories. This would produce a file called `asgn1.20220130.tar` in `~/mystash` containing directory `asgn1` and all files therein.
3. **(1 point)** Your script should be able to deal with both absolute and relative paths.
4. **(1 point)** If the script is not invoked with 2 arguments, it should print an error message, display the usage, and exit with **error code 1**.

   ```
   $ ./coderback.bash
   Error: Expected two input parameters.
   Usage: ./coderback.bash <backupdirectory> <fileordirtobackup>
   ```

5. **(1 point)** If the directory to store the tar file does not exist, the file or directory to back up do not exist, or if both arguments are the same directory, your script should print an appropriate error message indicating so, and exit with **error code 2** (do your checks in the order given in this question).

```
$ ./coderback.bash dir_does_not_exist file.txt
Error: The directory 'dir_does_not_exist' does not exist.
```

6. **(1 point)** If a tar file with the same name already exists, your script should ask the user whether they want to overwrite the file. If the user enters 'y' (lowercase letter Y), the tar file should be created, and the script should exit with **error code 0** (no error). Otherwise (for any other response), the tar file should not be overwritten, and the script should exit with **error code 3**.

```
$ ./coderback.bash $HOME/mystash/ asgn1
Backup file 'asgn1.200922.tar' already exists. Overwrite? (y/n)
```

Ex. 2 — Find the difference between directories (11 Points)

We will now work on a script that compares the text files between two directories, for example the source files in a directory to files with the same name in a backup directory.

1. Make sure that your script starts with a sha-bang to execute it using `bash` and is followed by a small comment section that includes your name, department and email id (format of this comment section is up to you). The script should also have additional comments for the important part of the code. **-1 point** if not followed.
2. **(7 points)** Create a script called `deltad.bash`. This script will take two directories as input parameters, iterate over the lists of files **(2 points)**, and report files which are either present in one directory but missing in the other **(2 points)**, or present in both directories but differ in content **(2 points)**.

```
$ ./deltad.bash $HOME/comp206/asgn2 $HOME/comp206/asgn2-bak
/home/20xx/csuser/comp206/asgn2/coderback.bash differs
/home/20xx/csuser/comp206/asgn2-bak/file1.txt is missing
/home/20xx/csuser/comp206/asgn2/file2.txt is missing
```

The actual ordering of files in the above output is not important. Your script MUST NOT consider subdirectories or files therein.**(-2 points)** if not implemented this way. The script should work with both relative and absolute paths. This also means the the output format displayed must contain a relative or absolute path depending on what the original argument was **(1 Point)**.

3. **(1 point)** If the script does not receive 2 input parameters, it should print an error message, display the usage, and exit with **error code 1**.

```
$ ./deltad.bash
Error: Expected two input parameters.
Usage: ./deltad.bash <originaldirectory> <comparisondirectory>
```

4. **(1 point)** The script should verify that both input parameters are directories, and that they are different directories. Otherwise, it should print an error message, display the usage, and exit with **error code 2**.

```
$ ./deltad.bash some_dir some_file.sh
Error: Input parameter #2 'some_file.sh' is not a directory.
Usage: ./deltad.bash <originaldirectory> <comparisondirectory>
```

5. **(1 point)** The script should function even if one or both of the input directories are empty.
6. **(1 point)** If no differences are found between the directories, the script should exit with **error code 0**. Otherwise (including missing files), it should exit with **error code 3**.

WHAT TO HAND IN

Upload both of your scripts, `coderback.bash` and `deltad.bash`, to MyCourses under the **mini 2** folder. You do not have to zip all of the files together. Re-submissions are allowed, but please try to upload all of the files again (and not just the modified ones) so that TAs do not have to go over multiple submissions to find correct files. You are responsible to ensure that you have uploaded the correct files to the correct assignment/course. There are no exemptions. **NO Emailing of submissions**. If it is not in MyCourses in the correct submission folder, it does not get graded.

Late penalty is -20% per day. Even if you are late only by a few minutes it will be rounded up to a day. Maximum of 2 late days are allowed. Even if you only (re)submit a part of the assignment (e.g., one script) late, late penalty is applicable to the entire assignment - No exemptions!.

COMMANDS ALLOWED

You may use any option provided by the following commands, even the ones that have not been discussed in class. Depending on your solution approach, you may not have to use all of these commands.

```
[[ ]]   $( )   $(( ))   $[ ]  basename
bc      break  case     cd    continue
date    diff   dirname  echo  exit
export  expr   for      grep  if
ls      pwd    read     set   shift
tar     while
```

You can also use redirection, logical and/or/negation/comparison/math operators and any check operators (such as checking if something is a file) as required with the `$[[ ]]` operator. You can also use shell variables and use/manipulate them as needed. You may use the concept of shell functions, but it might be an overkill for this assignment.
You may not use any commands that are not listed here or explicitly mentioned in the assignment description. **Using commands not allowed will result in 2 points deduction per such command.**

ADDITIONAL RESTRICTIONS

- Your scripts should not produce any messages/errors in the output unless you explicity produce it using an echo statement (following the message examples given in the question - also see the example tester output provided). Violating this would result in 2 points deduction per such message occurence.
- Your scripts should not take more than 10 seconds to run through the entire tester script / should not "hang", etc. This usually is a result of some logical error in your code. Any unfinished test cases will not receive points. To get an idea, a very naive solution for this assignment runs under 0.1s with the provided tester script.
- Your scripts must not create any files internally (other than what it is asked to produce as the final output tar file), even as tempoary output storage that the script deletes by itself. Doing this will result in 3 points deduction on the assignment score.
- DO NOT Edit/Save files in your local laptop, not even editing comments in the scripts. This can interfere with the fileformat and it might not run on `mimi` when TAs try to execute them. This will result in a 0. No Exemptions !!. TAs will not modify your script to make it work.

ASSUMPTIONS

- You may assume that you will have the necessary permissions as needed by the script for a correct invocation (e.g., it will not be executed on a file that you cannot read to archive, etc.)
- You can assume that that no file or directory names will contain any white space characters and will contain only alpha-numeric characters, underscore (_) and period (`.`) characters.
- For Ex2, you can assume that a file name in one directory will not show up as a directory name in the other directory.

MINITESTER

A tester script, `mini2tester.bash`, is provided with the assignment (along with an example output) so that you can test how your scripts are behaving.

It is recommended that you first run your scripts yourself, test each of the options and arguments using the examples above. Once you are fairly confident that your script is working, you can test it using the tester script. **DO NOT use the tester script before you have tested all of the options by yourself. If your script has too many issues, the output of the tester script could be overwhelming and discouraging.**

When you are ready, in order to run the tester, put the tester script in the same folder as your scripts for this assignment and run

```
$ ./mini2tester.bash
```

The idea is that the tester's output should be very similar or even identical to that of the example output provided, except for directory names.

TAs will be testing using a similar tester, with possibly different directory/file names but similar test case principles.

QUESTIONS?

If you have questions, post them on the Ed discussion board and tag it under mini 2, but do not post major parts of the assignment code. Though small parts of code are acceptable, we do not want you sharing your solutions (or large parts of them) on the discussion board. If your question cannot be answered without sharing significant amounts of code, please make a private question on the discussion board or utilize TA/Instructors office hours.

Please remember that TA support is limited to giving any necessary clarification about the nature of the question or providing general advice on how to go about identifying the problem in your code. You are expected to know how to develop a high level logic, look up some syntax/options and most importantly, debug your own code. Lab C covers a lot of useful debugging techniques. We are not testing your TA's programming skills, but yours. Do not go to office hours to get your assignment "done" by the TAs.

Emailing TAs and Instructors for assignment clarifications, etc., is not allowed. TAs and instructors may convert private posts to public if they are not personal in nature and the broader student community can benefit from the information (to avoid repeat questions). Also check the pinned post "Mini 2 General Clarifications" before you post a new question. If it is already discussed there, it will not get a response. You can email your TA only if you need clarification on the assignment grade feedback that you received.

FOOD FOR THOUGHT!

The following discussion is meant to encourage you to search independently for creative and optimal ways to perform rudimentary tasks with less effort and does not impact the points that you can achieve in the above questions.

- Can you include a third optional argument in your Ex2 script so that it compares only files with a specific extension?

  For example,

  ```
  $ ./deltad.bash $HOME/comp206/asgn2 $HOME/comp206/asgn2-bak sh
  ```

  would only compare the files in `asgn2` and `asgn2-bak` which end with the `.sh` extension.

  This option does not need to be included in the usage message, and will not be tested by TAs.

HINTS

Solving this assignment requires you exploring and learning a few small things on your own (in the spirit of the Unix culture). Some information is provided below to help you narrow down your search. You may not need all of these ideas - it largely depends on how you plan to solve the questions. You are not obliged to use them.

- Wondering how to extract the names of directory/file from a path? explore the commands `basename` and `dirname` to learn what they can do before starting to work on your assignment.

- How do you check if two arguments are pointing to the same directory? Just checking the arguments passed may not work as one could be absolute path and the other could be a relative path. There is an option you can use in the conditional operator (similar to the options used to check if something is a file, etc.). Explore to find that solution (not in the slides).
- Explore why and when the `tar` command displays the message *tar: Removing leading '/' from member names.* What option can you pass to the `tar` command to fix that and produce the same outcome as in the sample solution output?
- Explore the `-p` option to the `ls` command. What does it do? Is there a place in the assignment that you can use it?
- What option can you pass to `grep` to print only those lines that DO NOT match a pattern?
- Some times we are not interested in a message displayed by a command, only its intended outcome. Explore how the special file `/dev/null` can be used to discard unwanted messages.