School of Computer Science, McGill University

COMP-206 Introduction to Software Systems, Winter 2020

Mini Assignment 2: Bash Shell Scripting

Due Date Jan 30, 23:55

This is an individual assignment. You need to solve these questions on your own. Use the discussion forum on Piazza if you have any questions. You can also reach out to the course email address, utilize TA/Instructors office hours as necessary. Late penalty is -5% 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.

You MUST use `mimi.cs.mcgill.ca` to create the solution to this assignment. You must not use your Mac command-line, Windows command-line, nor a Linux distro installed locally on your laptop. You can ssh or putty from your laptop to `mimi.cs.mcgill.ca`, or you can go to the third floor of Trottier and use any of those labs to ssh to `mimi` to complete this assignment. All of your solutions should be composed of commands that are executable in `mimi.cs.mcgill.ca`.

Questions in this exercise requires you 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.

Please read through the entire assignment before you start working on it. You can loose up to 3 points for not following the instructions. It also has some helpful suggestions and tips that might save you some trouble.

Labs A through C will provide some background help for this mini assignment.

Unless otherwise stated, all the names of the scripts that you write, commands, options, input arguments, etc. are implied to be case-sensitive.

Total Points: 20

Ex. 1 — A basic shell script for searching tar files (8 Points)

In class we explored the tar command that can be used to archive files and entire directory structures. Sometimes we would want to check the contents of a tar file without extracting it back, such as to check if a particular filename is present in a given tar file. Programmers often keep such handy set of commands written together in the form of a shell script that can be used repeatedly.
For this task, you will create a shell script named `tarzan.sh` which accepts 2 arguments – the name of a file, followed by a tar file's name. The tar file name maybe passed on using absolute path or relative path. The objective is to inform the user if a particular file is present in a tar file or not, without extracting it. The `tar` command provides you with an option to list all the contents in the tar file without extracting it. Use the `man` command to explore the `tar` command and find this option.

Below is an example (valid) usage.

```
$ ./tarzan.sh  sort.c /archives/2017/gamingProject.tar
```

2 Points for writing code with proper (readability) indentation and writing necessary comments (required to understand the logic, etc.) in the code.

1.Use `vi` to create your script in `mimi.cs.mcgill.ca` .

2.Ensure that your script can be executed using `bash` shell.

3. **(2 Points)** If the script is not passed two arguments to it, you must throw an error/usage message, such as the following example.

```
$ ./tarzan.sh sort.c
Usage ./tarzan.sh filename tarfile
```

4. **(1 Point)** If the tar file cannot be found, the script should throw an error message.

```
$ ./tarzan.sh  sort.c /archives/2017/gamingProject.tar
Error cannot find tar file /archives/2017/gamingProject.tar
```

5. **(1 Point)** If the tar file is found, but the filename that is searched is not present in it, the script should display this fact.

```
$ ./tarzan.sh  sort.c /archives/2017/gamingProject.tar
sort.c does not exist in /archives/2017/gamingProject.tar
```

6. **(2 Points)** If the script is able to locate the filename in the tar file, it should display the found message.

```
$ ./tarzan.sh  sort.c /archives/2017/gamingProject.tar
sort.c exists in /archives/2017/gamingProject.tar
```

Ex. 2 — A bash script for file search and content display (12 Points)

Search is a very essential form of interaction with the contents stored in a file system. In this task, you will write a `bash` shell script `seeker.sh` that will look for file names with a specific string in the file name and optionally display its contents to the screen.
The usage syntax for the script is as follows.

```
$ /seeker.sh [-c] [-a] pattern [path]
```

The options/arguments in square brackets are optional, and their intent is as follows.
`-c` indicates that the script should display the contents of the file. Otherwise only the absolute path to the file (including file name) is displayed to the screen.
`-a` indicates that if there are multiple matching files, the output should include all of them. Otherwise only (any) one of the file is included in the output.
`path` is the absolute path to the directory under which the script should look for the files. If this argument is not passed to the script, it is expected to look for files under the "current directory".

`pattern` is a string that you want the script to look for in the name of a file (NOT contents of the file). For example if `pattern` is `msg`, it should match the file names `mymsg.txt, msg.txt, random.msg, lastmsgs.txt`, etc.
An example output of this script is as below.

```
$ /seeker.sh -c -a msg /mydata/scribbles
==== Contents of: /mydata/scribbles/msg2.txt ====
Hi, I am msg2.txt
Not all those who wander are lost.
==== Contents of: /mydata/scribbles/msg.txt ====
Hello there, I am msg.txt
```

2 Points for writing code with proper (readability) indentation and writing necessary comments (required to understand the logic, etc.) in the code.

1. Use `vi` to create your script in `mimi.cs.mcgill.ca` .
2. Ensure that your script can be executed using `bash` shell.
3. **(2 Points)** Ensure that your script is passed proper arguments. `pattern` is the only mandatory input to the script. An appropriate error and usage message should be raised if proper arguments are not passed into the script. See examples below.

```
$ ./seeker.sh
Error missing the pattern argument.
Usage ./seeker.sh [-c] [-a] pattern [path]

$ ./seeker.sh -c
Error missing the pattern argument.
Usage ./seeker.sh [-c] [-a] pattern [path]

$ ./seeker.sh -a
Error missing the pattern argument.
Usage ./seeker.sh [-c] [-a] pattern [path]

$ ./seeker.sh -c -a
Error missing the pattern argument.
Usage ./seeker.sh [-c] [-a] pattern [path]
```

4. **(1 Point)** If the directory passed as argument to the script does not exist, the script should display this fact.

```
$ ./seeker.sh msg /nosuchdir
Error /nosuchdir is not a valid directory

$ ./seeker.sh -a msg /nosuchdir
Error /nosuchdir is not a valid directory
```

5. **(1 Point)** If script cannot find any files with the specified pattern in its name, it should display this information.

```
$ ./seeker.sh -a meow /etc/cron.d
Unable to locate any files that has pattern meow in its name in /etc/cron.d.
```

6. **(1 Point)** When `-a` option is not passed, include only (any) one matching file in the output.

```
$ ./seeker.sh  msg /scribbles/messages
/scribbles/messages/msg2.txt
```

7. **(1 Point)** When `-a` option is passed, include all the files that match the pattern

```
$ ./seeker.sh -a msg /scribbles/messages
/scribbles/messages/msg2.txt
/scribbles/messages/msg.txt
```

8. **(2 Points)** When `-c` option is passed, contents of the files must be displayed

```
$ ./seeker.sh -c msg /scribbles/messages
==== Contents of: /scribbles/messages/msg2.txt ====
Hi, I am msg2.txt
Not all those who wander are lost.

$ ./seeker.sh -c -a msg /scribbles/messages
==== Contents of: /scribbles/messages/msg2.txt ====
Hi, I am msg2.txt
Not all those who wander are lost.
==== Contents of: /scribbles/messages/msg.txt ====
Hello there, I am msg.txt
```

9. **(2 Points)** When `path` argument to indicate the directory to look for files is not passed, search in the current directory.

```
$ cd /scribbles/messages
$ ./seeker.sh -c -a msg
==== Contents of: /scribbles/messages/msg2.txt ====
Hi, I am msg2.txt
Not all those who wander are lost.
==== Contents of: /scribbles/messages/msg.txt ====
Hello there, I am msg.txt
```

WHAT TO HAND IN

Turn in the 2 shell scripts `tarzan.sh` and `seeker.sh`, named properly (so that the TA can identify which script is for which question. You do not have to zip all of the scripts together.

MISC. INFORMATION

There is a tester script `mini2tester.sh` that is provided with the assignment that you can use to test how your scripts are behaving. TAs will be using the exact same tester script to grade your assignment.

```
$ ./mini2tester.sh
```

However, it is recommended that when you start writing your scripts, test it yourself first with each option and arguments such as the ones given in the above examples. Once you are fairly confident that your script is working, you can test it using the tester script.

You can compare the output produced by running the mini tester on your scripts to that produced by the mini tester on the solution scripts which is given in `mini2tester.out.txt`. Other than the names of directories used by the tester scripts, the remaining contents should match.

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.

- Will your script work for both

  ```
  $ ./seeker.sh -c -a msg
  ```

  and

  ```
  $ ./seeker.sh -a -c msg
  ```

  If not, can you think of a way to address it? Is your approach scalable and easy to maintain if we have three different options to pass? Does shell provide other efficient mechanisms to read options passed to a script?