School of Computer Science, McGill University

COMP-206 Introduction to Software Systems, Winter 2024

Assignment 2: Bash Scripting

Due Date Feb 12, 17:59 EST
TA Office Hours will start on Feb 6: See MyCourses announcement for list

Before you start, please read the following instructions:

- **Individual assignment**
  This is an individual assignment. You need to solve these questions on your own.
- **Use Discord and OH for questions**
  If you have questions, check if they were answered in the #clarifications channel on Discord or in the Discord q-and-a forum. If there is no answer, post your question on the forum. Do not post your code. If your question cannot be answered without sharing significant amounts of code, please use the TA/Instructors office hours. **Do not email the TAs and Instructors with assignment questions.** TAs should only be contacted by emails if you have questions about a grade you received.
- **Late submission policy**
  Late penalty is -10% per day. Even if you are late only by a few minutes it will count as one day. Maximum of 2 late days are allowed.
- **Must be completed on mimi server**
  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. You cannot use your Mac/Windows/Linux terminal/command-line prompt directly without accessing the mimi server. 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 doing parts of your assignment work elsewhere than the mimi server, you might lose all of the assignment points.** All of your solutions should be composed of commands that are executable in `mimi.cs.mcgill.ca`.
- **Must be completed with vim**
  A goal of the class is to get you to be comfortable in a command-line text editor. This wouldn't happen just from knowing the vim commands from slides: You become comfortable by using it a lot. Assignments are a good opportunity for that.
- **You must use commands from class**
  You may not use commands that you haven't seen in class. If you are not sure, ask on the Discord server.
- **Your code must run**
  For this assignment, you will have to turn in a shell script. 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. Your scripts should not produce any messages/errors in the output unless you explicitly produce it using an echo statement. Your script should run near-instantly: Hanging is usually a result of some logical error in your code. Your scripts must not create any files internally (other than what it is asked to produce as the final output tar file), even as temporary output storage that the script deletes by itself. DO NOT Edit/Save files outside of mimi, not even to edit comments in the scripts. This can interfere with the file format and it might not run on mimi when TAs try to execute them. **TAs WILL NOT modify your scripts in any ways to make them work.**
- **Please read through the entire assignment before you start working on it**
  You can lose up to 3 points for not following the instructions in addition to the points lost per questions.

Total Points: 20

In this assignment, you will be writing a `setup.sh` program that can perform a few different useful tasks. These are all common tasks that are used in real contexts and that you might use in your daily life as a programmer.

After doing this assignment, you will know how to...

- Run basic sequences of Unix commands using Bash.
- Write Bash if statements and loops.
- Make scripts that make your life easier as a programmer, for example making a backup or archive of your files.
- Make a script that can handle multiple tasks.
- Use wildcards to search for specific filenames.
- Add the result of a command to a filename and edit multiple filenames at the same time.
- Write a script that can handle relative and absolute paths.

Before you start, use vim to create a script called "`setup.sh`". Make sure that your script starts with a shebang to execute it using `bash` and is followed by a small comment section that includes your name and student ID. Remember that you might need to change permissions to be able to execute the script.

Ex. 1 — The different tasks (2 points)

Your `setup.sh` script will know which task to do based on the first command line argument it receives. There are three different tasks: `backup`, `archive`, `sortedcopy`. For example, if a user wants to use the backup task, they will write

```
$ ./setup.sh backup
```

Make an if-statement that will execute some code if the first argument is `backup`, some other code if it is `archive`, and some other code if it is `sortedcopy`. You will write the actual code in the exercises below: For now you can just put something like an empty echo statement that you will replace later. Note that some of the tasks will require more arguments: For now, you can ignore this.
If the first argument is not one of these three, or is not present, write this error message:

```
$ ./setup.sh
Error: Task must be specified. Supported tasks: backup, archive, sortedcopy.
Usage: ./setup.sh <task> <additional_arguments>
```

Ex. 2 — Simple backup (4 points)

When the user types `./setup.sh backup`, your script will copy all the txt files from a parent directory to a backup directory. It will also write down the date of the backup in a textfile. Scripts similar to this are very common and help keep track of your backups. The fact that it only selects txt files is for practice: In Ex. 3, you will see a more realistic version where the user can select the file extension.
For this exercise, the number of commands you should use for each step is specified. This is to make sure you get used to using the base Bash commands and do not use too many commands for simple tasks.
Inside of your if-statement case for `backup`, add the following:

1. **(1 point)** Print the absolute pathname of the current directory (where the script is ran from) and display the list of all `.txt` files inside of it using `ls` (don't use any flags for the command). This question should take two commands to implement.
2. **(1 point)** Create a `backup` subdirectory. Move inside of it. Display "Moved to backup directory", then display the absolute pathname of the backup directory. This question should take four commands to implement.
3. **(1 point)** Copy all `.txt` files from the parent directory to the `backup` subdirectory. Display "Copied all text files to backup directory". This question should take two commands to implement.
4. **(1 point)** Using redirection, create a file `date.txt` in the backup directory with the content "Current backup:". Then, using redirection again, append the current date and time using `date` (with no additional

flags or arguments) to the end of `date.txt`. Display `date.txt` using `cat`. This question should take three commands to implement.

Note that this is a simple script that makes the following assumptions (which we will also make when grading):

1. There are some `.txt` files in the directory you are running the script from. This means that when testing, you should create some `.txt` files in the directory where the script is ran from.
2. There is no file named `date.txt` in the directory you are running the script from
3. The `backup` subdirectory does not already exist. When testing your script, I'd recommend deleting the backup directory everytime by doing `rm -r backup` (be sure you do not delete a real backup directory that you previously had before this assignment though!)

Here are two additional tips to follow (also applies to following exercises):

1. The TAs might use testing scripts when grading your assignment. Make sure to display the exact text that the assignment asks for as the scripts might automatically compare your output to the expected output. For the same reason, use the same filenames.
2. Remember that the TAs will not run the script in the same directory than you do! Be sure that you do not hardcode any pathnames.

Ex. 3 — Archiving files (4 points)

The `archive` task will be similar to the previous task, with a few changes. Instead of creating a backup folder you will create a compressed archive. Archives are meant for long term storage: Unlike backup folders, the files will not usually be edited once an archive is done. The advantage is that they are smaller, and easier to move and share. Instead of having a date.txt file, the date will now be in the name of the archive itself. Another change is that instead of only copying txt files, the user will now specify the file format, as the second command line argument. For example, if the user wants to copy txt files, they will call `./setup.sh archive txt`

Add the following code inside your if-statement case for `archive`

1. **(1 point)** Check that a second command line argument was specified. Otherwise, output this error message

```
$ ./setup.sh archive
Error: Archive task requires file format
Usage: ./setup.sh archive <fileformat>
```

2. **(3 points)** Create a compressed archive `archive-YYYY-mm-dd.tgz` of all the files from the specified file format that are in the current directory. YYYY-mm-dd should be automatically replaced by the current date: You can use command substitution and look at the manual for the date command to see how to format the date as specified.
   When creating this archive, use the verbose option so we can see all the files that are archived. Display "Created archive `archive-YYYY-mm-dd.tgz`", and call `ls -l`

Note that you can assume that there are some files of the specified format in the directory.

Ex. 4 — Sorted directory copy (10 points)

The last task, `sortedcopy`, will copy all the files of a directory to a new directory, appending a number in front of each files so that they are sorted in reverse alphabetical order. Although this is a very specific task, the general idea of a script that loops through many files to rename them is very common and very useful. For example, TAs often use similar scripts to remove the text that MyCourses automatically adds in front of your filenames when they download it!

`sortedcopy` will require the user to specify two more command line arguments: the source directory, and the target directory.

As this is a longer script, you should also have additional comments for the important parts of the code. You may lose points if this is not followed.

Add the following to your if-statement case for `sortedcopy`:

1. **(2 points)** As mentioned above, your script will take two additional inputs:
   - The name of an existing directory whose files will be copied. This will be referred to as the "source directory".
   - The name for the new directory where files will be copied (Will be overwritten if it exists, see part 3 below). This will be referred to as the "target directory".

   Note that the script should work with both relative and absolute paths.

If the script is not invoked with 2 additional arguments, it should print an error message, display the usage, and exit with **exit code 1**.

```
$ ./setup.sh sortedcopy
Error: Expected two additional input parameters.
Usage: ./setup.sh sortedcopy <sourcedirectory> <targetdirectory>
```

The script should verify that the second input parameter is a directory. Otherwise, it should print an error message, display the usage, and exit with **exit code 2**.

```
$ ./setup.sh sortedcopy some_file.sh some_dir
Error: Input parameter #2 'some_file.sh' is not a directory.
Usage: ./setup.sh sortedcopy <sourcedirectory> <targetdirectory>
```

2. **(7 points)** Your script will create the target directory and copy all files from the source directory into it, appending "#." in front of each filenames, where # is the position of the file in the original directory in reverse alphabetical order.
For example, if

```
$ ./setup.sh sortedcopy documents sorted_docs
```

is called, and the source directory `documents` has the following files

```
bravo.txt
echo.jpg
tango.gif
```

Your script will create the target directory `sorted_docs` and copy the files such that the content of `sorted_docs` is:

```
1.tango.gif
2.echo.jpg
3.bravo.txt
```

Finally, the script will terminate with **exit code 0** upon success.
Note that the script should function even the source directory is empty, **-1 point** otherwise. You should also only copy files from the source directory, you should not copy subdirectories and the files they contain. **-2 points** otherwise. For example, if you had a directory "my_dir" in the source directory from earlier:

```
bravo.txt
echo.jpg
my_dir
tango.gif
```

The target directory's content would still be:

```
1.tango.gif
2.echo.jpg
3.bravo.txt
```

, regardless of the presence of my_dir in the source directory and whatever files that subdirectory might contain. Once again, make sure to check that your program works with both absolute and relative paths.

3. **(1 point)** If a directory with the same name as the target directory already exists, your script should ask the user whether they want to overwrite the directory. If the user enters '`y`' (lowercase letter Y), the conflicting directory should be deleted and replaced by the new empty directory of the same name. Otherwise (for any other response), the directory should not be overwritten, and the script should exit with **exit code 3**.

```
$ ./setup.sh sortedcopy some_dir sorted_dir
Directory 'sorted_dir' already exists. Overwrite? (y/n)
```

WHAT TO HAND IN

Upload your script, `setup.sh`, to MyCourses under the **Assignment 2** folder. You do not have to zip the file, but you might need to if MyCourses does not accept the file format. Re-submissions are allowed, but note that TAs will only grade the most recent submission. **You are responsible to ensure that you have uploaded the correct submission.**