Shell Scripting 2020 second week

Format of the learning diary and scripts

The tasks are returned in moodle.

Return the learning diary in **PDF format**. Preamble the document with the following information:

- Name and student number

The learning diary should contain the answers to questions made in the "Put in your answer" parts. Also example output can be put into the answer.

Include the code in week's directory as a tarball with scripts named with the task name as "task#.sh", i.e. for task 2 the name would be "task2.sh".

Deadline for this set of tasks is Monday 16th of November 2020 and I will release the next set of tasks on 9th of November.

If you run into problems, try to read the man pages and try google. If you are still stuck with the task send me a msg in moodle and I will answer during normal office hours (9-17).

Pipelines (9)

We are now ready to try a more advanced technique, connecting one program's stdout to a second program's stdin. This is achieved through the use of the | character, aptly named the pipe in shell programming. The result is called a pipeline (http://en.wikipedia.org/wiki/Pipeline_(Unix)).

By doing /bin/ls ~ you'll get a list of files in your home directory. By connecting the output of ls to wc -l, you can count the number of files in any directory you choose. Now try /bin/ls ~ | wc -l.

Put in your answer:

- Now redirect the output of ls to a file and then count the number of lines in the file (HInt: man tee). Show that you get the same number of lines using both methods.
- Modify the command to count only the number of folders in your home directory.

Filters (10)

By using multiple pipes, you can combine several programs into a single pipeline. Each command can then filter the output of the preceding command.

The grep command is arguably one of the most powerful filters. It is used to select lines by matching them against a regular expression.

For example, doing ls ~ | grep 'e' will list only the files in your home directory whose names contain the letter 'e'. The match is case-sensitive.

Cool hint: to get color-highlighted results from grep, use grep --color=auto. Oh, maybe an alias would be useful?

Put in your answer:

- Pipeline ls to grep 'e' to wc in order to count the number of matching files. Present your solution and its output.

Interlude: Bash (11)

The bash shell is similar to wc -l and ls in the sense that it reads stdin from the keyboard and writes stdout to the terminal by default. In reality, bash is a command interpreter, similar to java, ruby, and many others.

You can print out the location of the bash program by doing which bash. In almost all Unix-like (http://en.wikipedia.org/wiki/Unix-like) systems, including Linux, the location is by convention /bin/bash.

There are other interpreters than bash, but we will concentrate on bash on this course for simplicity. Thus, our shell script files begin with the line

```
#!/bin/bash
```

by itself. This is an instruction for this interpreter to run /bin/bash and let that interpreter read and execute the lines following #!/bin/bash.

The #! is called a sha-bang. Yes, really.
http://www.tldp.org/LDP/abs/html/sha-bang.html#FTN.AEN201

From this point onwards, *whenever I say write a shell script that does (...)*, you will begin by

1. Creating a text file. Sometimes the name is given, many times not. Naming the file sensibly is very important for code re-use, so try to invent descriptive names. There is no real harm in making them lengthy. Moreover, it is advisable to add purpose of

this script comment after the sha-bang where more details, such as the usage, options etc., about the script can be given. Example: touch ls-homedir.sh
2. Giving the text file execute permissions. Example chmod u+x ls-homedir.sh
3. Opening the text file and writing #!/bin/bash on the very first line

Let's try that now.

Put in your answer:

- Create a bash script count-homedir.sh which counts the number of files in your home directory. Verify that it works. Attach the script contents to your report.

```
In the future, you will usually begin by copying (cp) the script file that most
closely matches your currently problem. Don't just rewrite one file, it's good
practice to keep the old versions too. So keep a version of this week's script in
your week folder Week2.
```

Some Assembly required (12)

Remember the Insert date -exercise from Week 1? Good. Let's continue building on that.

Recall that the Exactum-kamera keeps track of past images as files in subdirectories.

Put in your answer:

- Write a bash script that lists all files and subdirectories from November 2011. Use ls.

```
Hint: ls does not list subdirectory contents by default. It's called going into
subdirectories recursively. Check either man ls or ls --help.
```

Now we want only the pictures (13)

Put in your answer:

- Write another script that skips the subdirectories from the previous exercise. In other words, it only lists the picture files ending in .jpg.

```
Hint: Use grep as a filter
```

How many do we have so far? (14)

Put in your answer:

- Write a script that counts the number of picture files.

```
Hint: Pipe output to grep and then to wc -l.
```

Remember the backticks (15)

Put in your answer:

- Combine backticks (`) and the date command from the exercises of Week 1 to list the number of image files of the current month. In this case, it's now November.

```
NOTE: There seems to be something broken with the camera so you cannot get any images
for current month but the command is what is asked. So you can use older dates

Hint: You can verify that your script works correctly by comparing its output to the
output of the solution of the previous exercise.

Hint 2: There is a modern variant of the backticks, $(  ). The excellent BashFAQ has
the rationale. http://mywiki.wooledge.org/BashFAQ/082
```

The big brother of ls (16)

The previous examples are somewhat naive solutions, but they really work. In some cases, ls will no longer be sufficient for the problem at hand.

Luckily, it has somebody to call for backup. That program is called find, and it is much more powerful than ls.

find takes it's input a bit differently. See man find for details, but most often you may be using the program like find /path/to/search -expression1 -expression2 [...]. (Don't try to execute that, it won't work.) For example, find ~/Desktop -type fwill find and print the names of each 'normal' file in the directory ~/Desktop, if it exists in your home directory.

Put in your answer:

- Combine everything. Write a script that finds the image files corresponding to the current month and counts them. Use find, date, and wc -l.