CS 50 Software Design and Implementation

Lab2

Shell Programming Lab

This lab comprises five bash shell programming problems. They start easy and get progressively harder.

Note: This lab and all future labs will be graded.

You will find that some of the snippets of shell script in Lecture 4 on shell programming can be used as a foundation for some of the problems.

These exercises should develop your ability your shell programming skills. Grading will focus on the correctness of your solutions, but will also consider good coding style - including consistent formatting, selection of identifier names, and use of meaningful comments.

Submitting assignment:

We are using SVN for the submission of assignments. Create a directory for each new lab (viz. lab1, lab2, lab3, lab3, lab5, lab6 and lab7). We will make a copy of the directory after the deadline.

Please make sure that each lab directory (e.g., lab2) contains a simple text file (called README) briefly describing the source code in the directory and anything “unusual” about how your solution should be located, executed, and considered. Essentially, your README gives us a quick overview of the content on the directory and how to run your programs.

Your svn repository root is at `https://svn.cs.dartmouth.edu/classes/cs50-S12/yourreponame`. For example: `https://svn.cs.dartmouth.edu/classes/cs50-S12/campbell`. Note, repo in yourreponame is short for respository.

But replace campbell with your repsoitory account username. You should have received this from Wayne or the TA. When you click on the svn URL above you will be asked to enter a username and password: enter your full DND (also blitz name) as your username and Blitz password as your password; DO NOT use your CS account name and password to login to the svn server - it will not work.

NOTE, for classes after 2012 this is relevant: change cs50-s12 to the correct year and term for example W13 changes the svn commands below to cs50-W13.

All detailed svn instruction is here:http://www.cs.dartmouth.edu/~campbell/cs50/svn.html.

Coding style: Please put comments in your scripts to help increase its understanding. Include a header in each script similar to the the one in spy.sh.

```
# Script name: spy.sh
#
# Description: Monitors when users log in and out of a machine and sends e-mail on
# each time a user logs in and out to the user. It also sends a summary and the end of
# the day to all users informing them of their usage patterns including how many times
# they logged in and out and the duration. The script also computes which user logged
# in the most often, and for the longest and shortest periods of time.
#
# Input: List of users to monitor in terms of their full names such as ‘‘Andrew Campbell’’
#
# Output: E-mails the user each time they log out. And sends a e-mail summary of activity
# at the end of the day to all users monitored.
```

It is important when your write scripts or C programs to program **defensively**. That is, you need to check the program's input and provide appropriate warning or error messages back to the user in terms of the program usage. For example, a script that requires input arguments should check if it has the right number of input arguments and if not it should inform the user of usage errors.

Complete the following questions.

---

1) **birthday_match.sh script**: Write a script called birthday_match.sh that takes two birthdays (or in the general case any two dates in history) of the form MM/DD/YYYY (e.g., 05/15/1959) and returns whether there is a match if the two people where born on the same day of the week (e.g., Friday). Do you know which day you were born on? Should be fun to see if any of you are born on the same day of the week. The input and output to/from the script should follow this format:

```
# no match
```

```
./birthday_match.sh 12/01/1984 12/02/1984
The first person was born on: Sat
The second person was born on: Sun
Therefore, you are not born on the same day.

# match

./birthday_match.sh 11/25/1984 12/02/1984
The first person was born on: Sun
The second person was born on: Sun
Jackpot: You were both born on the same day!
```

Please code defensively for the cases of no arguments or insufficient arguments.

2) **leap_year.sh script**: Write a script called leap_year.sh that takes a year as input and determines if its a leap year. If the user does not enter a correct leap year then the script should compute the next leap year after the date entered as an argument. The script command line and its response should look like the following:

```
# incorrect entry

./leap_year 201
201 is not a leap year.
The closet leap year after that year is 204

# correct entry

./leap_year 2008

2008 is a leap year.
```

Here is the definition of a leap year. You will find **pseudo code** for the algorithm to compute a leap year there.

Your code should be defensively written to handle incorrect entry, such as no arguments entered or too many. In each case the user should be given an appropriate response for example:

```
./leap_year
leap_year usage: No arguments entered, please enter a year.
```

3) **count_linesinfiles.sh script**: Write a script called count_linesinfiles.sh that counts the number of ordinary files (not directories) in the current working directory and its sub-directories. For each ordinary file found your script could count the number of lines in the file and print the total number of lines for files in the directory tree rooted at the current working directory.

Hint: The unix command find without any arguments is a good place to start.

There are no input arguments to the count.sh script. The expected output is shown below.

```
./count.sh
Counting lines of all files...
23 ./count.sh
1  ./a.java
24 Total
*************************
Number of files found: 2
*************************
```

4) **wget_search.sh script**: Write a script called wget_search.sh that given a text file (containing a list of URLs) and a sequence of words, searches for occurences of the words in the webpages. This script has some relationship to the function of a search engine's crawler. We will look at the design and implementation of a simple command-line based search engine later on in the course.

You should use the **wget** command to retrieve the html pages from the web URLs in the file. The wget command is a non-interactive network downloader. Each retrieved web page should be stored and renamed using a progressive integer number (e.g., 1.html, 2.html, 3.html and so on).

The command has the the following format:

```
./wget_search.sh url.txt word1 word2 ... wordn
```

```
where:

- url.txt is the name of the file containing the list of URLs - one per line (e.g, http://www.dartmouth.edu/);

- word1 word2 ... wordN are the words to be checked (e.g., computer).
```

The number of words is not known a priori. The program should be case sensitive.

For example, given the following list contained in the file url.txt:

http://www.cs.dartmouth.edu
http://www.dartmouth.edu

The execution of the command returns the following output:

```
./wget_search.sh url.txt Dartmouth Campbell

Dartmouth
http://www.dartmouth.edu 11
http://www.cs.dartmouth.edu 0

Campbell
http://www.dartmouth.edu 0
http://www.cs.dartmouth.edu 0
```

5) **spy.sh script**: Write a script called spy.sh that checks if any users given as input to the script, e.g.:

```
$ ./spy.sh "Andrew Campbell" "Xiaochao Yang"&
```

are logged on to a machine (in this case wildcat). When the user logs out the script sends the user an e-mail with the subject title

“Gotcha username!” The e-mail should include the time the user logged in and out, and the duration of time on the machine (e.g., 20 mins).

Update 2014: Because CS50 is a large class we will use a number of machines and split the class into groups. The machines are:

moose, pierce, goatisland, sabattus, halfmoon, merced, hancock, wildcat, and kinsman.

We will discuss the breakdown of groups to machines in class.

In Lecture we used a script to find the username from the /etc/passwd file. One simple way to see if the user is running a process is to use the unix command who. There are other ways too.

Important: Andrew and the TA will each log on to wildcat a number of times on **Sunday until 10 PM** - they’ll sneak on, your mission if you accept it is to take them down. To allow you to test your script Andrew and the TA will sneak on to wildcat on **Saturday** too. The TA will post what we did on Sat. to see if you caught us.

Extra credit: Just after **10 PM Sunday** send an e-mail to Andrew and the TA that lists the following:

```
Dear Sneaky cs50 Guys:

Caught you sneaking on to wildcat today. Here’s a summary of my surveillance:

Andrew you logged on N times for a total period of X. Here is the breakdown:

1) Logged on Sat Jan 12 18:40:34 EST 2008; logged off Sat Jan 12 18:40:34 EST 2008
2) ...

TA you logged on Z times for a total period of X. Here is the breakdown:
1) Logged on Sat Jan 12 18:40:34 EST 2008; logged off Sat Jan 12 18:40:34 EST 2008
2) ..
3) ..

Looks like Andrew || TA the  spent most time on wildcat today - 100 mins in total for
all his sessions;  Andrew || TA was on for the shortest session for a period
of 2 mins, and therefore the most sneaky; and, Andrew || TA logged on the
longest session of 15 mins.
```

```
(Note, three variables need to be  maintained by spy to be able to correctly fill in the paragraph above correctly --
- i.e., Andrew OR TA. This is a note and should not be included in the email).

Thought you could sneak by my code hey  - nailed you.

Best,

Your name

Note, in the above we use the TA title but you'll have to use the TA's name of course.
```

Some General Tips: Starting and Killing Deamons, Debugging Scripts

How do I run a script?

You need to make sure the script which you create using an editor (e.g., vim or emacs) is excutable; by default its a plain text file. Assuming I've just written spy.sh and saved it. Now make let's it executable:

```
$chmod +x spy.sh
$ ls -l spy.sh
-rwx------ 1 campbell faculty 8136 2008-01-21 14:11 spy.sh
```

Now we can run spy.sh.

How do I run my dastardly spy.sh daemon process (in background)?

First our spy.sh process must be written in a "continous loop" with a delay at the end of the loop. The loop in the process means it will run forever. We want spy to sleep for 60 seconds and then wake up and start spying. The delay is important. Just think of it as my job security ;-). OK, so we have written spy as a "while [true] loop" with a sleep 60 and now we want to launch it in the background so that when we log out of wildcat the daemon spy process remains running, forever.. so it thinks. Our spy program wouldn't be much good if it was killed when we log off the machine, right?

Here is how we start our spy.sh in background.

```
$ ./spy.sh "Andrew Campbell" "Xiaochao Yang" &
[1] 9070
campbell added to spy list
shu added to spy list
```

From the above command line you can see we use "&" to push the spy.sh into background OK let's see if it's running using ps

```
$ ps
  PID TTY          TIME CMD
 8118 pts/9    00:00:00 bash
 9070 pts/9    00:00:00 spy.sh
10368 pts/9    00:00:00 sleep
10398 pts/9    00:00:00 ps
```

Note, that we can not only see the spy.sh process but also the sleep command excuted as part of the spy.sh script. Why is this? Recall I mentioned in class that the shell creates a new process for every command we execute unless it's a built-in. Most of the time our spy.sh will be sleeping. It checks things then sleeps .. ad nauseam. So it's likely that the next time you do a ps we will see two processes running again: the spy.sh and its associated "sleep 60" command. Let's check if that is the case:

```
$ ps
  PID TTY          TIME CMD
 8118 pts/9    00:00:00 bash
 9070 pts/9    00:00:00 spy.sh
11221 pts/9    00:00:00 sleep
11253 pts/9    00:00:00 ps
```

Yes.

How do I kill my spy deamon?

If you start your spy script and then log off wildcat it will still be running. You want that to be the case because you do not know when I or the TA will log on and off of wildcat. You can assume we will stay on longer than 60 seconds (why is that important) so your spy will always catch us.

During the debug phase you will be revising and testing your spy script. So you will want to "kill" your perviously started spy before launching your new one. Note, last year some students unbeknowns to themselves started 10s of spys and never killed them! Not good. Note, that your "while [true] loop" has an **mail command** in it - this is particularly **dangerous**. Imagine you launch a deamon that is in a tight loop (e.g., your sleep is missing from the code) mailing everyone on the machine (because your who awk filter is incorrect) . Multiply that by 5 because you didn't kill your previous buggy deamons. The result is that you tie up the machines processor (problematic, but no big deal), bring down the department email server and fill up people's inbox with zillions of gotcha mail. The result: Campbell is shipped back to the UK. So please take care with this spy program when you are debugging, I like living here.

Here is how we kill your deamon spy. Assuming we start the spy in background on wildcat and then log off. When we log back on again to wildcat to carry on the assignment and do a ps we see:

```
$ ps
  PID TTY          TIME CMD
12373 pts/11   00:00:00 bash
12412 pts/11   00:00:00 ps
```

There is no spy running!

Well there is but it is not associated with the shell that created it (or the parent process ID to be accurate) so we do not see it. But we need to be able to see it. Here is one way to find it. If we do a "ps -el" we see all the processes running on wildcat. And, if we use grep we can see all the spys running.

```
$ ps -el | grep -i spy
0 S   529  9070     1  0  80   0 - 26396 wait   ?        00:00:00 spy.sh
```

When you do this on Sunday you will see all the students spy scripts running. So if you want to just see the spy deamon you created (and importantly, all the spys you created and not killed, if that is the case) then type:

```
$ echo $UID
529
$ ps -el | grep -i 529
0 S   529  9070     1  0  80   0 - 26396 wait   ?        00:00:00 spy.sh
5 S   529 12372 12368  0  80   0 - 24308 poll_s ?        00:00:00 sshd
0 S   529 12373 12372  0  80   0 - 26988 wait   pts/11   00:00:00 bash
0 S   529 13479  9070  0  80   0 - 26161 hrtime ?        00:00:00 sleep
0 R   529 13487 12373  0  80   0 - 26364 -      pts/11   00:00:00 ps
0 S   529 13488 12373  0  80   0 - 25684 pipe_w pts/11   00:00:00 grep
```

By first gettting your user ID (mine is 529) and then using ps and grepping the output for all processes created by user 529 (me in this case) we see the spy deamon and a bunch of other processes that belong to me not visible when just using ps by itself.

Now that I can see my spy deamon I'm going to "kill it". I'm not going to worry about the sleep process because when the sleep 60 is done and wakes up it will find its parent has been killed (its parent process is spy which executed the command sleep 60). A child process (sleep in this case) with no parent (in this instance) will die - it's an orphan to use the Unix vernacular I hate all these negative vibes that Unix gives but that is how it is; the 70s were clearly a brutal time for OS designers.

To kill the spy deamon we need the PID (process ID). From the above pipeline output we can see the UID (USER ID) followed by the PID. The PID of my spy is 9070.

The final Shakespearean act now falls on us:

```
$ kill -9 9070
```

Let's make sure its killed:

```
$ ps -el | grep -i 529

5 S   529 12372 12368  0  80   0 - 24308 poll_s ?        00:00:00 sshd
0 S   529 12373 12372  0  80   0 - 26988 wait   pts/11   00:00:00 bash
0 S   529 13859     1  0  80   0 - 26161 hrtime ?        00:00:00 sleep
0 R   529 13867 12373  0  80   0 - 26365 -      pts/11   00:00:00 ps
```

```
0 S   529 13868 12373  0  80   0 - 25684 pipe_w pts/11   00:00:00 grep
```

No spy.

And if I wait for another 60 seconds and do a ps

```
$ ps -el | grep -i 529
5 S   529 12372 12368  0  80   0 - 24308 poll_s ?        00:00:00 sshd
0 S   529 12373 12372  0  80   0 - 26988 wait   pts/11   00:00:00 bash
0 R   529 13907 12373  0  80   0 - 26364 -      pts/11   00:00:00 ps
0 R   529 13908 12373  0  80   0 - 25683 -      pts/11   00:00:00 grep
```

No sleep process. All cleaned up. Now I can safely start my new spy again.

Simple Debugging Tips.

When you run a script you can use printf or echo to print out places in your script where the excution reaches e.g., “echo Got here” or print out the contents of script variables as a sanity check: for example, in spy you need to maintain a number of arrays one of which is needed to determine if users are logged in - echo $USERS˙LOGGEDIN[$USERCOUNT].

Another tip: if the script give you a syntax error; for example:

```
$ ./count.sh
./count.sh: line 13: syntax error near unexpected token ‘done’
./count.sh: line 13: ‘done’
```

When testing.

On the Saturday that we test your program we will sneak on to the machine. You can check the times that campbell (and the TA/section leader) logged on to the machine by using the last command. You can use this to verify the times that your spy.sh program computed. This is useful as a sanity check.

```
$ last campbell
campbell pts/8        c-71-192-136-118 Sun Jan 19 22:39   still logged in
campbell pts/5        c-71-192-136-118 Sun Jan 19 22:07 - 22:12  (00:04)
campbell pts/5        c-71-192-136-118 Sun Jan 19 22:00 - 22:04  (00:04)

wtmp begins Sun May 29 15:49:36 2011
```

The error is on or around line 13. In emacs edit the file ./count.sh again and then “goto line 13” using the sequence of key strokes “ESC g g” that is hit the ESC key and hit g twice. Then, enter the line number 13 and you will be brought to that line. Now fix the bug.

OK. Hope these tips help you.

Final tip: Make sure you always logout when you are done and see the prompt to login again before you leave the terminal.