**Regis University CC&IS**

**CS465 Unix**

**Lab Homework #3**

**MATTHEW HARTIGAN**

**29-MARCH-2019**

\*answers included below in red\*

Complete the following 17 exercises on a Unix system. Fill in your answers on this document. Points that will be given for each correct answer are listed by each question.

### See the end of the document for submission instructions.

### Initial Instructions:

### These exercises will be performed in the Bourne shell. But your default shell is the Korn shell. So before you begin, start a Bourne sub-shell by typing: sh

### When you are DONE with your homework, type: exit

### to exit the Bourne sub-shell. This will return you to your original Korn shell.

### NOTE: If you logout and return later, be sure to start a Bourne shell again, before proceeding with your homework.

### Exercises:

1. (4 points) Create two files using the **echo** command and output redirection. The first file, **labdata1**, should contain the line:  
            File 1 data -- line 1  
   The second file, **labdata2**, should contain two lines:  
            File 2 line 1  
       Second line

List ALL the commands that you used to create the files.

Since the instructions say to only use echo command and output redirection, here is one such answer:

echo File 1 data – line 1 > labdata1

echo -e “File 2 line 1\nSecond line” > labdata2

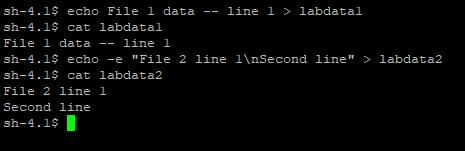
To create labdata2 using echo, I had to use the -e option with quotes so the command would accept the newline special character, which made the input line cramped in my opinion. A different way you could create labdata2 without using echo could be as follows:

cat << STOP> labdata2

File 2 line 1

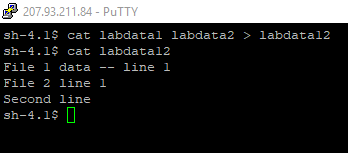
Second line

STOP



1. (2 points) Create a third file, **labdata12**, by concatenating **labdata1** and **labdata2** and redirecting the results to **labdata12**.           What command did you use?

cat labdata1 labdata2 > labdata12

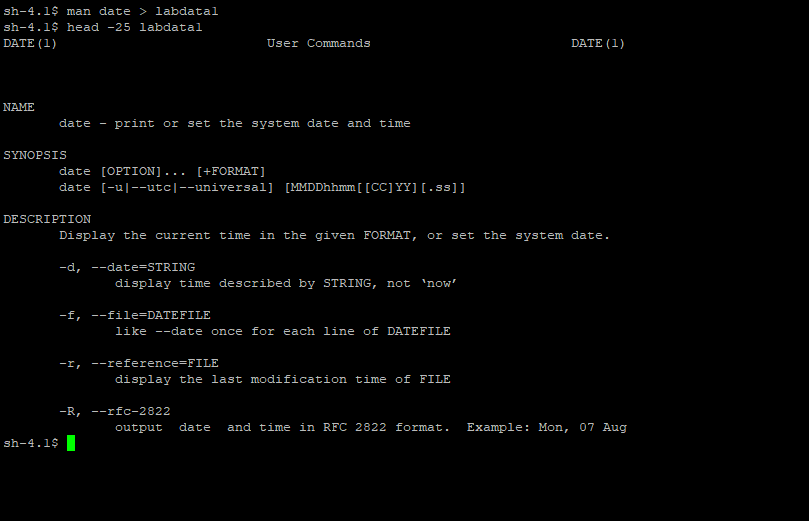


1. (2 points) By default, **stdout** is displayed on the screen. Run the command to display the manual page for the **date** command, and redirect the **stdout** of the command to a new file, **labdata1**.

          What command did you use?

Assumption: We are starting this task in the same working directory we were in when we completed task #2 with no further changes. Therefore, labdata1 is a file that already exists in the directory.

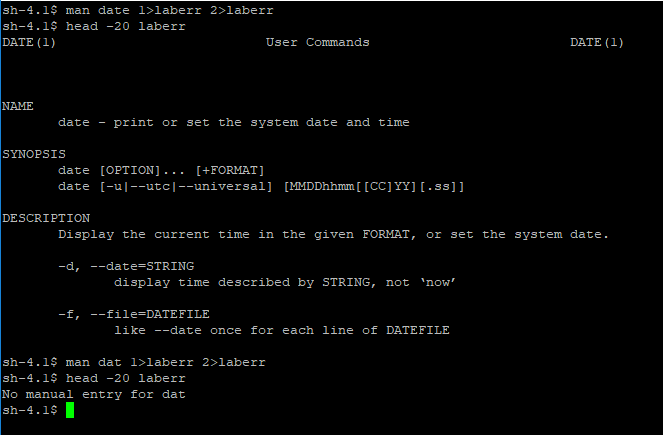
man date > labdata1



1. (2 points) Display **labdata1** again. What happened to the original contents of **labdata1**?

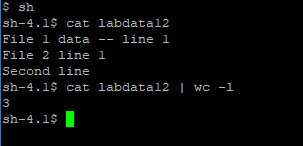
The file labdata1 now contains the contents of the manual page for the date command. Because labdata1 already existed before we input the command shown in task #3 (and we have write permission for labdata1), the original contents got overwritten once the output from the new command was redirected to the file.

1. (2 points) By default, **stderr** is also displayed on the screen. Display the manual page for the **date** command, and redirect **both** the **stdout** and **stderr** of the command to **laberr**. (To confirm this works, try typing the command incorrectly to generate an error).  
             What command did you use?  
     
   man date 1>laberr 2>laberr



1. (2 points) Pipe some commands together to display **only** a count of the number of lines contained in the **labdata12** file.          What command did you use?

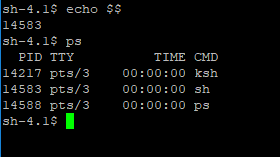
cat labdata12 | wc -l



1. (2 points) Observe your shell prompt. What is it? What does it tell you about your shell?

sh-4.1$

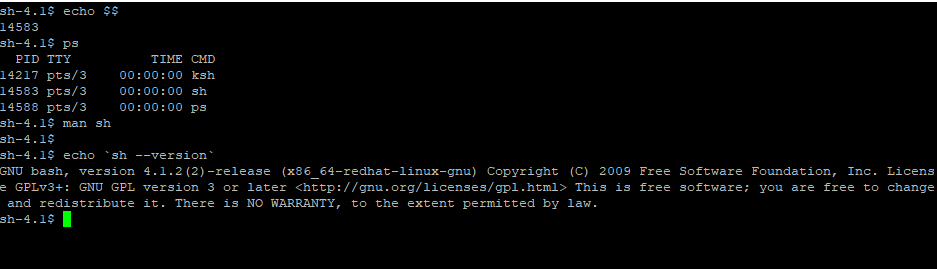
After we logged into the Unix system and launched a Korn shell, we then launched a Bourne subshell using the command sh. Our prompt now shows that we are currently in the Bourne subshell, version 4.1. We can confirm this by looking at the processes that are currently running (we see that the current process, 14583, matches the number of the sh subshell process).



One other point to make- I wanted to verify that the 4.1 shown in the command prompt did correspond to the version number of the Bourne shell implementation we are using. When I looked (echo `sh –version`), I was surprised to discover that sh is a link to a Bash shell, version 4.1.2.

The online content and textbook readings this week (as well as several online sources) explain that Bash is an implementation of the Bourne shell, so it is okay to use to complete our assignment this week.

<https://stackoverflow.com/questions/5725296/difference-between-sh-and-bash>



1. Change your prompt:

*NOTE: Make sure you are in the Bourne shell, or these commands will not work correclty*

* 1. (2 points)

Type: **PS1='\d \t $ '**

Explain what happened to your prompt.

The command prompt now displays the date, time, and $ character. This is because the above command changes the shell variable $PS1 to contain this information instead of the \s and \v information that is the system default for the sh command prompt.



* 1. (2 points)

Type: **PS1='$PWD> '**

Type: **cd ..**

Type:  **cd**

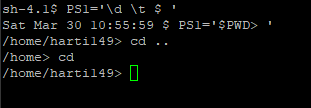
Explain what happened to your prompt.

In the first command for this subtask, we redefined the $PS1 shell variable. This time we redefined it again to hold the value of the $PWD (pathname of the current working directory) instead of the date and time from subtask a. This changed our command prompt accordingly to the current working directory of /home/harti149> as show in the snip below.

cd .. takes us up one directory, so the working directory and the command prompt both change to /home as shown in the snip below.

cd takes us back to the user home directory, and the command prompt changes to match once more.

In short, if we use this command prompt setting, the prompt will always match the directory you are currently in (useful!).



You may type: **PS1='\s-\v\$ '** to restore the original Bourne prompt, if you wish.

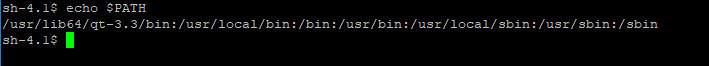
1. Display the value of your PATH environmental variable.   
   1. (2 points) What command did you use?

echo $PATH

* 1. (2 points) What was displayed (the exact output)?

/usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin

See second line of below snip.



* 1. (1 point) Is your current working directory part of the path?

The user home directory (/home/user) is the current working directory, but it is not currently a part of the path because it is not shown in the echo $PATH command.

* 1. (2 points) If it was not, how would you add it to your path?

PATH=$PATH:.

Note that the above command will adjust the PATH variable so that whatever the current working directory is, you are able to execute files from it.

1. (3 points) What do you need to do to a script file, after you create it in **vi**, before you can run it? If your script file was named **script1**, what ***exact*** command would you use to do so?

You need to change its permissions to make it executable. You can choose to set it to be executable by the user, group, and/or other. In this case, we will show the command to add execute permissions for only the user.

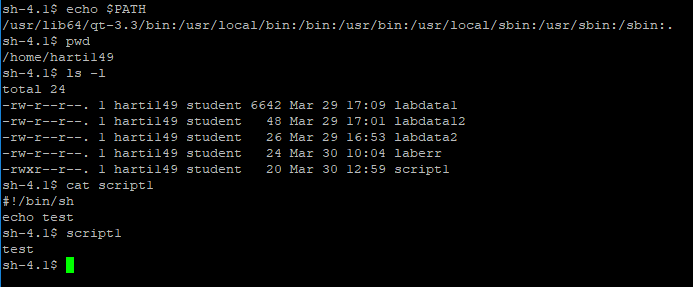
chmod u+x script1

1. (2 points) What command would you use to run a script in your current working directory called **script1**?

We assume that we already added the ‘.’ to the path in task #9d and added executable permissions to script1 in task #10. Therefore, we would only need the following command to run a file called script1 in our current working directory:

script1

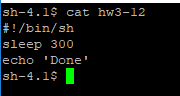
The proof of this is shown in the below snip.



For questions 12-17, include the **contents** of **each script file you create**, as well as answering any questions asked (the script contents can just be placed into this document). Do NOT just include a screen shot – include the actual ***text*** commands in each script file.

1. Create a **Bourne shell script** called **hw3-12** that runs the **sleep** command for 300 seconds, then outputs the word "Done" to the screen.
   1. (2 points) What commands are in your script file?

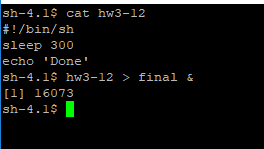
script for hw3-12:



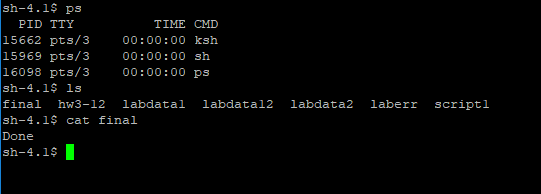
b) (2 points) Run the script in the background, redirecting the output to a file called **final**.

What command did you use?

hw3-12 > final &



After the ‘sleep 300’ command finishes, we can verify that the output was correctly redirected to ‘final’ by opening the file and displaying the contents. See below snip.



Next you will view your running processes and then terminate the script process before it completes.

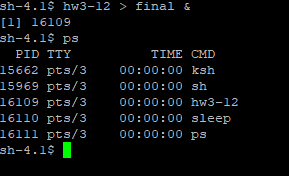
c) (2 points) What command did you use to view your running processes?

Note that for tasks c, d, and e in this section, we will restart the hw3-12 script in the background so there are process(es) ongoing for us to terminate.

ps

d) (1 points) What was the ***exact*** output of the command?

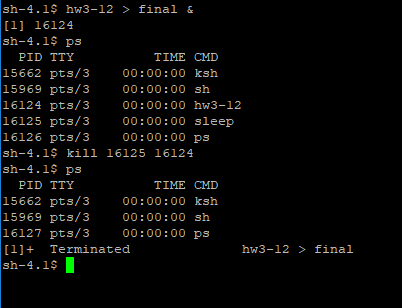
See below screenshot for exact output.



e) (2 points) What command(s) did you use to terminate your background process(es)?

Note that first we restarted hw3-12 as a background process. In the below screenshot, it was assigned PID 16124 and its subcommand ‘sleep’ was assigned PID 16125. Therefore, to kill both process(es), we can use the command:

kill 16125 16124



1. Enter the following commands at your shell prompt (***not*** within vi) to create a script file called **hw3-13**:

Type: **cat > hw3-13**

Type: **echo $1**

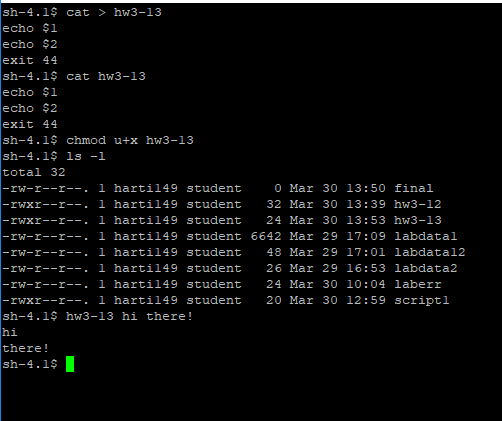
Type: **echo $2**

Type: **exit 44**

Press: **CTRL-d**

View the script by typing: **cat hw3-13**

Make the script executable, and then run it by typing: **hw3-13 hi there!**



a) (3 points) What does your script output?   
Explain why it displays each line of output.

The script outputs the following (as shown in above snip):

hi

there!

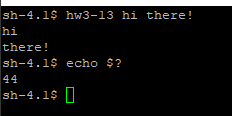
When we run the script with the command: ‘hw3-13 hi there!’, hi and there! are arguments related to the command that are stored as shell variables. hi is positional parameter $1, and there! is positional parameter $2. Because these arguments are shell variables, the hw3-13 script can access them while it runs. the first echo command: ‘echo $1’ displays positional parameter $1 to standard out (the screen). Hence, we see ‘hi’ on a line by itself. The second call to echo: ‘echo $2’ similarly displays positional parameter $2 to standard out.

Immediately after running the script, type: **echo $?**

b) (2 points) What is output? Explain why that value was output.

The output is (see below snip):

44



$? represents the exit status of the most recently executed command. When we echo $? immediately after running the hw3-13 script, we get an output of 44 because the script itself sets the exit status of 44 in its last line (‘exit 44’).

1. (11 points total, allocation noted below)

Write and execute a Bourne shell script called **hw3-14** that will

a) (6 points) Display your **username**, your **process ID**, the name of your default **login shell**, and your **home directory** to the screen, along with identifying output statements.

See below.

b) (2 points) Display a list of all users who are currently logged in, along with an identifying output statement.

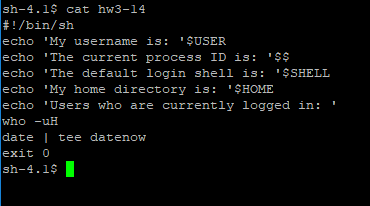
See below.

c) (2 points) Use one piped command (NOT 2 commands separated by a semi-colon) to display the **date** both to the screen and saved to a file called **datenow**.

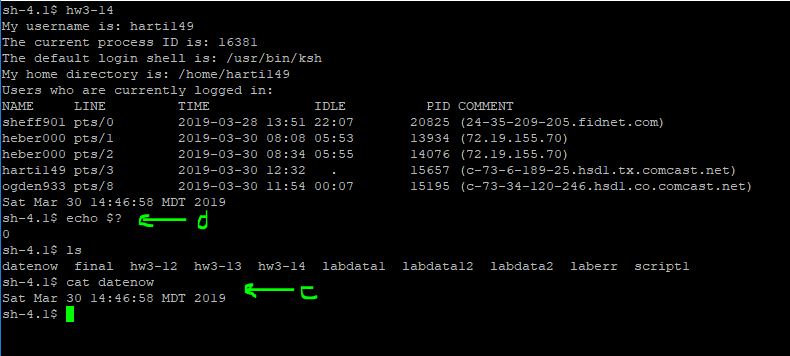
See below.

d) (1 points) Exit with an exit value of 0.

See below.

hw3-14 script file  


output validating hw3-14 script file:



1. (15 points total, allocation noted below)

Write and execute a Bourne shell script called **hw3-15** that will

From within the script, create three **background** processes:

a) (2 points) one that saves a long listing of your *hidden* files to a file named **hiddenlist**

See below.

b) (2 points) one that lists your processes status in a file named **pstat**

See below.

c) (2 points) one that stores only the count of the number of words from the file **hw3-13** into a file named **wordcount**

See below.

The script should then:

d) (1 point) issue a command to **wait** for ALL three background processes to complete.

See below.

After the background processes have all completed, the script should:

e) (1 point) Display a simple message that they are done. See below.

f) (2 points) Ask whether to:

**display** the contents of the files that were created

OR

**delete** the files that were created

See below.

g) (2 points) Read in the user's choice See below.

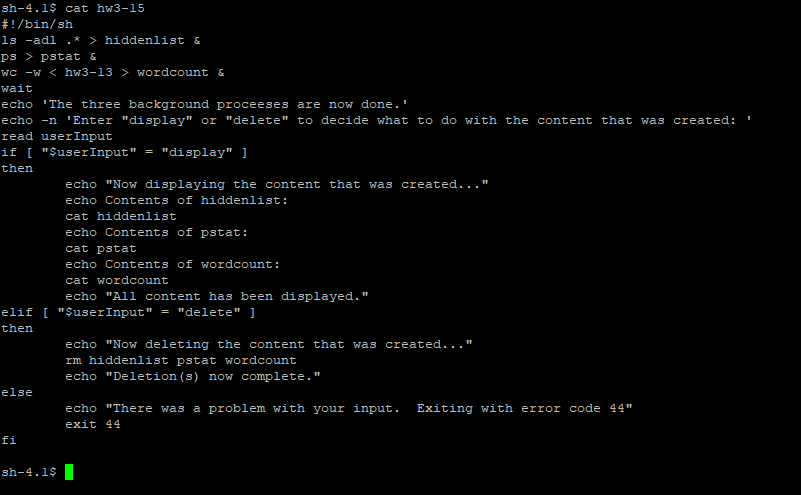
h) (4 points) Perform whichever the user chooses to do

(NOTE: if the user chooses delete, display a message confirming deletion

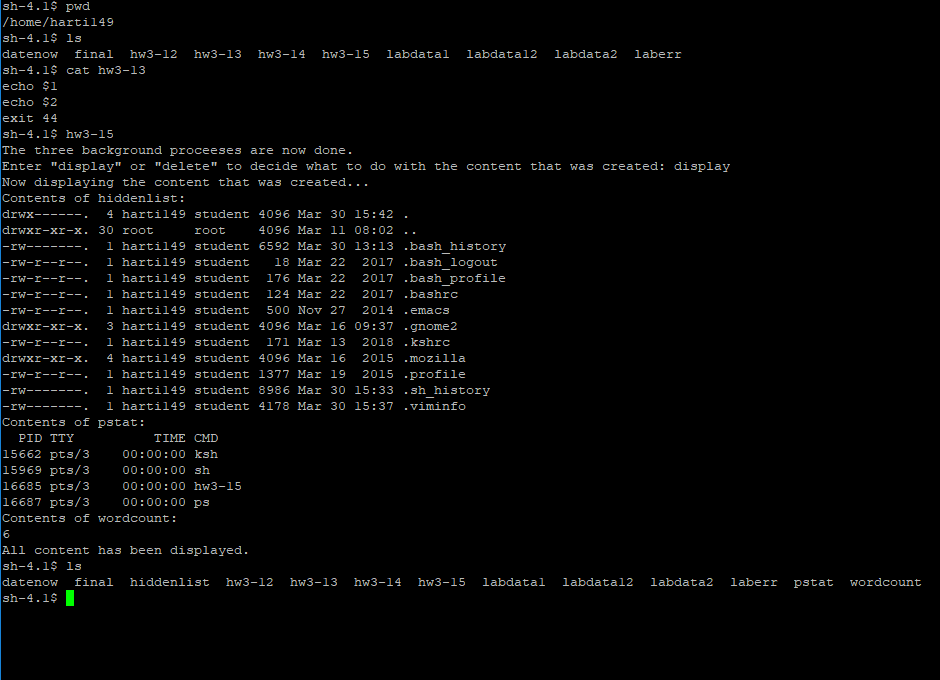
of the files before exiting).

See below.

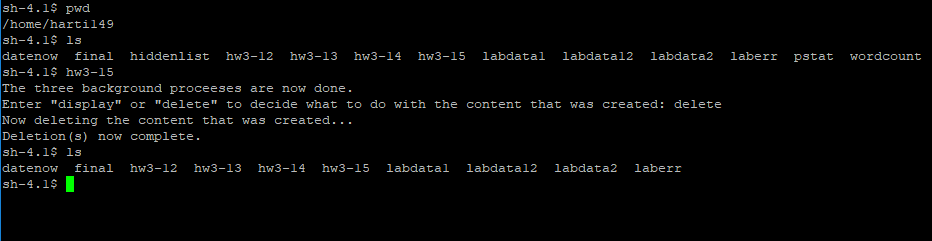
hw3-15 script file



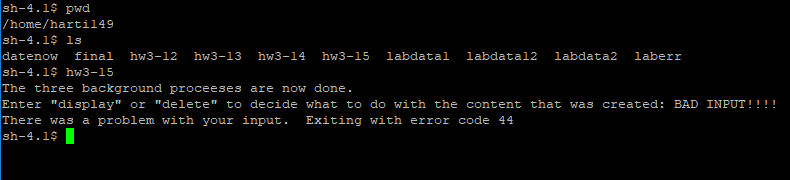
output validating hw3-15 ‘display’ case:



output validating hw3-15 ‘delete’ case:



output validating hw3-15 ‘bad user input’ case:



1. (13 points total, allocation noted below)

Write and execute a Bourne shell script called **hw3-16** that will:

a) Have two **command line arguments**. The first argument is a **number** and the second argument is a **name**. See below.

b) (3 points) The script should start by checking the number of command line arguments entered. It should display an **error message** if exactly 2 arguments are NOT given on the command line. See below.

c) (2 points) If two arguments were given, the script should display a message that says **Processing [scriptname]**... where scriptname is the name of the script (using one of the built-in variables, NOT hardcoded). See below.

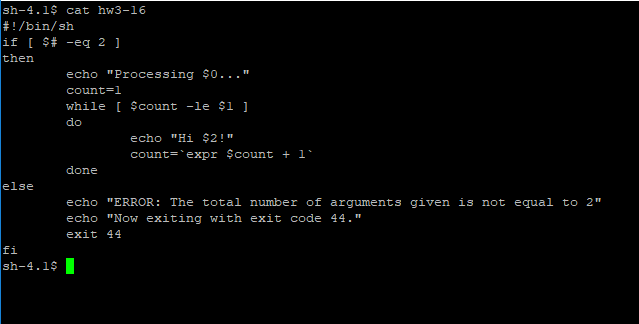
d) (8 points) It should then display a greeting and the name to the screen, number times: See below.

For example, the command:  
 $ hw3-16 2 Mary  
would produce the output:

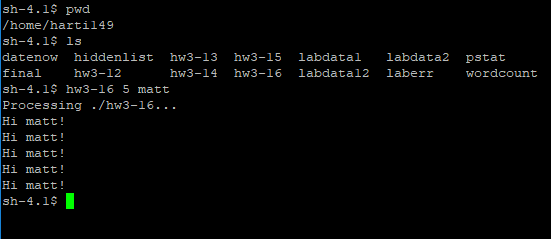
Processing /home/user/hw3-16...

Hi Mary!  
 Hi Mary!

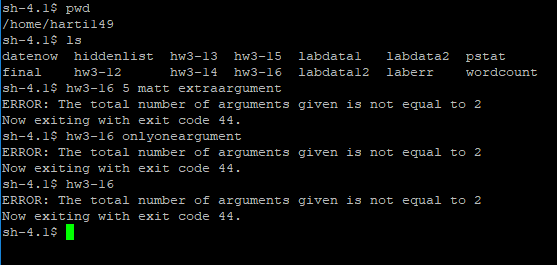
hw3-16 script file



output validating hw3-16 standard input case:



output validating hw3-16 bad input cases:



1. (16 points total, allocation noted below)

Write and execute a Bourne shell script called **hw3-17** that:

Requires a **list of numbers** as command line arguments. The script will sum the odd arguments. The script should:

* + 1. (4 points) First display a **count** of **all** arguments:

Example: There are 5 arguments.

But if no arguments are listed when the script is run, the output should just be:

     No arguments to process.

and the script should exit. See below.

* + 1. (10 points) Examine each the number that was given as command line argument and determine if it is an odd number. If it is odd, display it and add its value to the odd number sum. (Hint: Try using a **for-in** loop) See below.
    2. (2 points) After all the command line arguments have been examined, display the **sum** of the **odd** arguments. See below.

For example, the command:  
           $ hw3-17 2 53 87 34 96 11

would produce the output:

There are 6 input arguments.

Odd arguments:  
 53

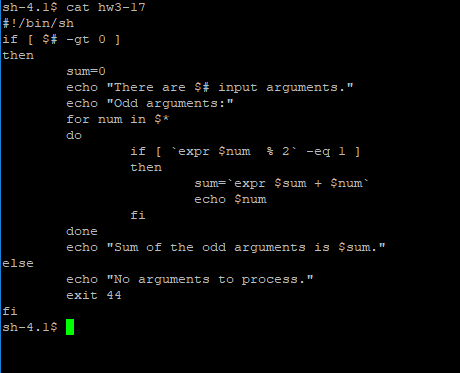
87

11

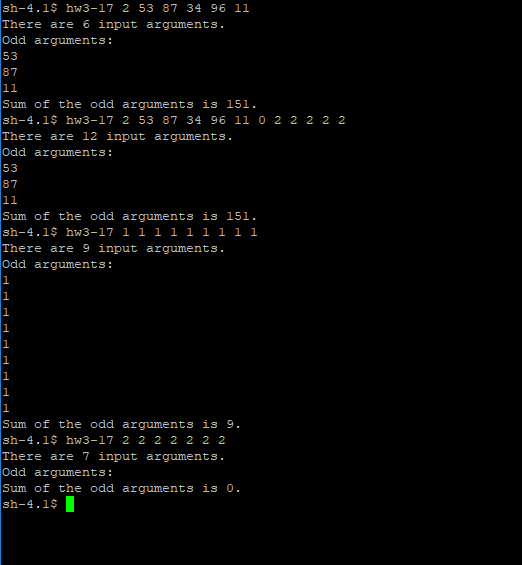
Sum of the odd arguments is 151.

Note: If there were arguments, but none were odd arguments, the sum displayed should be zero.

hw3-17 script file



output validating hw3-17 input cases:



**Submission**

This homework assignment is due by midnight Sunday (last day of Week 3).

Submit a filled in copy of this Word document to the **Homework Assn 3** drop box (located under the Dropbox tab in the online course).

Before submitting the Word file with your answers, you MUST rename it as follows:

### Lastname-hwk3.docx

For example:

### Smith-hwk3.docx