CSCI 275 - UNIX Scripting

**Homework 02**

CH 10

1. Explain the difference between "$@" and "$\*" including an example of where (how/why) you would use each of them.

“$@” essentially creates an array of the arguments. Let’s say we have a script containing the following commands:

for arg in “$@”

do

echo $arg

done

If we run the program by doing **./testFile red blue green yellow**, the output will be each color on its own line. This is because the $@ splits the input into separate arguments. So, each word would be its own argument. On the other hand, the “$\*” essentially creates a string of all the arguments, separated by a space, or the default IFS. So, if we continue with the same script and arguments, the output would be all the colors on the same line, separated by a space. For example, if you have a script that requires a quote as input, the quote being any quote the student picks from their role model, if the script uses the command **quote=$\***, the input will be saved as a string in the variable **quote**. But if you have a script that will save your grocery list, you could use the command **groceries=$@** to save your list of groceries as an array in the variable **groceries**.

2. Rewrite the journal script of Chapter 8 (exercise 5, page 377) by adding commands to verify that the user has write permission for a file named journal-file, and that it exists, and if the user does not have write permission to the file exit with a standard fail error code. If there was an error, display the error message to standard output and standard error. If the script completes correctly, exit with the status code for success.

#!/bin/bash

# journal: add journal entries to the file journal-file

echo Running $0

#Test if file exists

#If file doesn't exist, exit code is 1

echo

if [[ ! -e $1 ]]

then

echo "ERROR: This file does not exist" 1>&2

echo "ERROR: This file does not exist"

exit 1

fi

#Test if file is writeable

echo

if [[ ! -w $1 ]]

then

echo ERROR: You do not have write permissions for $1

exit 1

fi

echo

date >> $1

echo -n "Enter name of person or group: "

read name

echo "$name" >> $1

echo >> $1

cat >> $1

echo "-------------------------" >> $1

echo >> $1

echo

echo Finished running $0

3. Write a script that takes a comma-separated list of items and outputs the items, one per line, to standard output without the commas.

#!/bin/bash

echo Running $0

for arg

do

newArg=${arg%,}

echo $newArg

done

echo Finished running $0

4. Develop a shell script named Fibonacci that generates a “Fibonacci-like” sequence of numbers, delaying between the printout of each number to the screen.

If you are not familiar with the math behind the Fibonacci numbers, read: <http://en.wikipedia.org/wiki/Fibonacci_number>

In pseudo-code, the non-recursive version of the Fibonacci algorithm is:

Seed Value1

Seed Value2

Loop:

Sum = Value1 + Value2

Output Sum

Value1 = Value2

Value2 = Sum

End Loop

In general, you are going to take two numbers as input, add them together to get a third number, which is added to the second number to obtain a fourth, which is combined with the third. Note: You are not writing a program that only produces “the” Fibonacci sequence. It will produce the sequence if the two provided numbers are 0 and 1 (or 1 and 1).

Your script should meet the following criteria:

* You should use “getopt” to process the options provided on the command-line. By default, the delay between each number output should be 1 second, but this should be configurable.
* The two seed values should be provided on the command line, with options. (Use the getopt and its options parameters - not just taking the last two provided parameters.)
* The number of values to be printed before exiting can be provided as an option. If not provided, the script should continue until interrupted.
* If the script is not started with enough parameters, you should provide a standard usage message and exit with a failure exit code.
* The script should print out the number of values that were calculated and printing as the last item before exiting – both during normal completion and interruption.

The completed script should be uploaded from your account at a lab machine using $ submit 275 Fibonacci