CSE 390, Autumn 2012
Assignment 5: Basic Shell Scripting
Due Tuesday, October 30, 2012, 12:30 PM

This assignment focuses on Bash shell scripting at an introductory level. Electronically turn in files **`.bash_profile`**, **`mantra.sh`**, and **`filestats.sh`** as described in this document. To receive credit, your `.bash_profile` should also be present in your home directory with proper permissions.

Task 1: Login script, .bash_profile (revisited)

As we have discussed, `.bash_profile` is a script that runs every time you log in to a Bash shell. For this part of this assignment, add to your previous `.bash_profile` file in your home directory to perform the following new operations:

1. *(Self-Discovery)* Use the `mesg` command to set whether or not you want to be able to be contacted by commands such as `write` and `wall`. Either setting (yes or no) is fine; but explicitly set one or the other.

2. Set the user's prompt to consist of the current username, shortened host name, and working directory, with separators between each. For example, if user `bob` is logged into host `attu3.cs.washington.edu` and is in directory `/usr/bin`, the prompt would be:

```
bob@attu3:/usr/bin$
```

Use appropriate commands and/or environment variables to discover each piece of info (Who is the current user? What host are they on?) to insert into the prompt. A tricky part is inserting the current directory; you can't do this by just running the `pwd` command, because it won't update each time the user changes to a new directory. Instead, insert the special symbol `\w` into your prompt text to tell Bash to put the working directory into the prompt.

3. At the end of the file, output this message to the user as he/she logs in. For user `bob`, the message might be:

```
Hello, bob!
Your shell is /bin/bash.
The current time is 01:43 PM, Monday April 26, 2012.
```

To be clear, you are not supposed to literally output `bob` and exactly that date/time shown above. You are to use appropriate commands to discover the current user's username and to display the current time in the format shown. Read the `man` pages about displaying the current date/time with an appropriate `"+FORMAT"` parameter. For example, to show a date such as `04/26`, you could write:

```
date "+%m/%d"
```

(Self-Discovery) If you want an optional extra challenge worth 0.001 points of extra credit, change the first output line above to print the user's real name rather than username.

```
Hello, Marty Stepp!
```

To do this, you would need to perform a lookup on the person's real name based on their username. You can do this by examining the contents of the `/etc/passwd` file for the given username. There should be a line like this:

```
bob:x:1000:1000:Robert Paulson,,,:/home/bob:/bin/bash
```

Notice that the various information about the user is separated by colon `:` characters. The `cut` command can chop a line into fields based on a delimiter and select only certain field(s) to output. You can use it twice: once to separate the tokens by colons, and a second time to remove commas after the name. You may assume that the format of these lines always contains the same number of colon-separated tokens in the same order.

Shell scripts

Put a **comment header** atop each of your script files indicating your name, course, etc. and a description of the program. Each of your two scripts should also have a proper `#!` heading so that it can be used as an executable file.

For reference, our `mantra.sh` is 26 lines (14 "substantive" lines in the CSE 142 Indenter tool) and our `filestats.sh` is 21 lines (13 "substantive"). You don't need to match these totals exactly or be close to them; they are just a ballpark.

Task 2: Shell script, mantra.sh

For this exercise, write a shell script file `mantra.sh` that accepts **two command-line arguments**: a string for a message to print, and a number of times to print it. The script should print the message that many times, surrounded by a box of stars. For example, if the user runs your script in the following way:

```
./mantra.sh "All work and no play makes Jack a dull boy" 5
```

Then the script should produce the following output to the shell:

```
******************************************
* All work and no play makes Jack a dull boy *
* All work and no play makes Jack a dull boy *
* All work and no play makes Jack a dull boy *
* All work and no play makes Jack a dull boy *
* All work and no play makes Jack a dull boy *
******************************************
```

Notice that you can pass a multi-word message by putting it in quotes. (Bash handles that for you.)

You may assume that the user will pass 2 parameters and that their values will be reasonable; the message will be non-empty and shorter in length than the width of your terminal, and the number of times will be a valid integer greater than 0.

Hint: Perhaps the trickiest part of this script is getting the lines of stars above and below the message to be exactly the right length to surround the messages. You can use an appropriate command to find out the length of the message and surrounding stars, then print a star this many times above/below. Use `echo` and `` ` ` `` backticks to combine commands.

Task 3: Shell script, filestats.sh

For this exercise, write a shell script file `filestats.sh` that accepts an arbitrary number of file names as command-line arguments and prints information about each of the files:

- number of **lines** in the file
- number of **blank lines** in the file, and **percentage** of blank lines out of the total lines
- number of **characters** and **words** in the file, and approximate number of **characters per word**

(computed as the truncated integer division of total characters divided by total words)

The web site file `hw5.tar.gz` contains some test files. If you decompress it to the current directory and then run:

```
./filestats.sh Jack.java lyrics.dat
```

Then your script should produce the following output to the shell:

```
Jack.java:
  lines: 13
  blank: 3 (23%)
  chars: 369 in 44 word(s) (8 char/word)
lyrics.dat:
  lines: 27
  blank: 2 (7%)
  chars: 873 in 180 word(s) (4 char/word)
```

Note that the shell expands command-line arguments containing wildcards before your script runs. For example, if your directory has files `file1.txt` , `file2.txt` and `file3.txt` and you run `filestats.sh *.txt`, the `$@` array doesn't contain the string `"*.txt"`. Instead it will contain three elements: `"file1.txt"` , `"file2.txt"`, and `"file3.txt"`. You may assume that each file listed as a command-line argument will be readable by your script. You may also assume that no file will be empty (0 lines, 0 characters). But a file might contain 0 blank lines or might be arbitrarily large/small. If no arguments are passed to your script, it should produce no output.

Hint: It can be tricky to figure out the number of blank lines in a file. You can see which lines of a file are *non*-blank by using the `grep` command and looking for lines that match the pattern `"."` (a dot). This doesn't literally look for a period character; a dot represents any character in general. (We'll learn more about this when we cover "regular expressions.")