Unix Scripting Lab

Part 1: Shell Review

1. Create a new directory for the course in your home folder called `lab`. Inside, create the following files:
    - `main.c`, `game.c`, `enemy.c`, `hero.c`, `a.out`
    - `monster.h`, `human.h`
    - `.highscore`
2. Create the following directories under `lab`
    - `Music`, `Misc`, `Drivers`
3. Display all files starting with an `e`
4. Copy all files and folders starting with a capital letter to a new directory called capitals
5. Delete all files whose extension is a single letter
6. Rename both occurences of `Misc` folder to `Test`
7. Delete all files containing m

---

1. List all files containing a lowercase letter in their name, AND the nonexistant file named `Hidden`
2. Now show the same list, but redirect standard output to a file
3. Now show the same list, but redirect standard error to a file
4. Combine 2 and 3: Redirect standard output to one file, and standard error to another
5. Create 3 files: `file1`, `file2`, `file3`
6. Use `hostname` to write the current host name into `file1`
7. Prevent file clobbering
8. Repeat (6). Did you get an error ?
9. Fix the error keeping the noclobber option set

Part 2: Environment

1. Create a new directory named: `I have $5`
2. Create an alias that finds all files larger than 2k but smaller than 5k
3. Create an alias that finds all directories in /tmp owned by the current user
4. Create an alias that finds all files modified within the last 4 hours
5. Create a shell function that finds partial matches of a file name, so you could type: `findpartial txt` to get all files with txt in their name
6. Create an alias for `cp` that turns it to `cp -i`
7. Create an alias for `rm` that turns it to `rm -i`
8. Create an alias that prints how many files exist under current directory

9. Create an alias that prints how many executable files exist under current directory
10. Create a shell function that takes a date and prints how many files were modified in that date

Part 3: Getting Parameters

1. Write a shell script that takes a file name as input and prints the file backwards
2. Write a shell script that takes two file names as inputs, and replaces their contents.
3. Write a shell script that reads a file name from the user, prints its contents and the number of lines in the file.
4. Write a shell script that takes a Windows file (lines end with `\r\n`) and converts it to a Unix file (lines end with `\n`).

Part 4: Conditionals

1. Write a shell script that takes an input argument and tells if it's a string or a number (Hint: try `expr a + 0`)
2. Write a shell script that takes 3 input arguments and prints out the largest one
3. Write a shell script that reads a name from the user - if that name is an executable program run it, otherwise print its content. If it's not a file print an error message.
4. Write a shell script that takes two file names, and prints the contents of the larger one.
5. Write a shell script that asks the user for a number, if the user chooses 7 - print "You Win".
6. Write a `safedel` script. The script takes a file name as command line input, and moves that file to a `~/TRASH` directory instead of deleting it.
   Upon invocation, script should check `~/TRASH` for files older than 48 hours and delete them (hint: use `find`).
7. Write a shell script that reads a file name from the user, checks that the file is valid, and lowecases its name. For example, running `lc MyFile` should rename the file `MyFile` to `myfile`.

Part 5: Loops

1. Write a shell script that takes input as command line arguments and prints them out backwards (first argument printed last).
2. Write a shell script called "wait_for_user" that takes a user name and checks if the user is logged in. If she's not logged in, the script sleeps for 5 seconds and checks again in a loop - until the user logs in.
3. Write a shell script that reads a file and prints its content double-spaced (adding a blank line after each line)
4. Write a shell script that reads a file and prints its content with no blank lines.
5. Write a shell script that reads a file and prints out only the longest line
6. Write a shell script that takes a two file extensions as input (call them ext1 and ext2),

and renames all files ending with ext1 to end with ext2.
7. write a shell script that takes several file names as inputs, and copies itself to each of the files. don't forget to set execute permissions on the target files.

Part 6: Named Pipes

1. Create a named pipe called `bob`
2. Print out the list of files to the named pipe. Notice ls blocks.
3. Read the contents of the pipe using cat. Notice ls unblocks.
4. Write a shell script that creates a named pipe and listens on it. For every new line it reads from the pipe, it should create a new file whose name is the first word in the line. Can you delete the named pipe when the script ends ?
5. Write a shell script that creates a named pipe and then executes `find /`. If it reads the word "exit" from the named pipe, it should stop the find and quit. Hint: `$!` is the process id of the last started process

Part 7: Functions

1. Write a shell function called sum that returns the sum of its arguments
2. Write a shell function called countExecutables() that takes a directory name as parameter and returns the number of executable files in that directory.
3. Write a shell function that prints out the multiplication table. Function should take a number `n` and print a table sized `n*n`. For example, running `mul 5` should produce:

```
    1   2   3   4   5
1   1   2   3   4   5
2   2   4   6   8   10
3   3   6   9   12  15
4   4   8   12  16  20
5   5   10  15  20  25
```

1. Write a shell script that includes the following functions:
2. `add_contact` takes a name and an email.
3. `list_contacts` prints out a list of all available contact details and emails
4. `email_contact` takes a name and some text, and sends the text to the contact's email address (as specified before when the contact was added).

Use a contacts.txt file to store the data.
Now write another script which uses the functions

Part 8: Sed

1. Add a blank line after each line of input
2. Change an existing file, so each line should start with a '> '
3. Use sed to perform the following two replacements:
4. If a line starts with `#`, replace each character with a `-`
5. For all other lines, replace each character with a `.`
6. `sed = filename` prints out the file with line numbers. Use another sed in a pipeline to join each number to its line (removing the newline).
7. Emulate head with sed (print top 10 lines)
8. Emulate `tail -1` with sed (print last line of a file)
9. Emulate `uniq` with sed (delete consecutive duplicate lines)
10. Delete duplicate words from input line
11. Replace the first and last word in every line