CS 618 Lab-1

Part 1: Typing

Why? The faster you type, the faster you code. We do not want to spend any conscious energy on typing. All your conscious energy can be spent on thinking about the code that you are writing.

- Learn touch typing from: https://www.typingstudy.com
- Do one lesson everyday. With regular practice, you shall be able to come to 40 to 60 words per minute. You may regularly measure your typing speed here: https://www.typingstudy.com/speedtest

Part 2: Editing

Why? You will need to code in a variety of environments such as by logging into a remote server or into an embedded device. GUI-based editors may not be available there. Touching the trackpad/mouse and clicking buttons is very slow.

- Learn vim. https://github.com/iggredible/Learn-Vim

Part 3: Shell scripting

Why? You can save a lot of time writing one-off programs if you are comfortable with basic shell scripting.

Note: It's best to work in a Linux environment with bash. Same commands in other shells like zsh (default on Mac) may have different semantics. You can know your shell type by running: echo "$SHELL"
We will use a linux machine for all the evaluations. Thus we recommend testing your scripts on a linux based machine at least once.

Commands to learn: ls, cd, rm, mv, cp, touch, top, pwd, mkdir, ln, cat, echo, grep, awk, sed, parallel, cut, sort, uniq, wc, head, less, xargs, ping, nmap, ssh, find, tree

You can run `man <command>` to learn about a command.

Some resources:

1. http://linuxcommand.org/lc3_learning_the_shell.php

2. https://ryanstutorials.net/bash-scripting-tutorial/
3. https://www.geeksforgeeks.org/bash-script-arithmetic-operators/

Question 1 (Count Files)

Count the number of text files starting with a given prefix within a given directory.
Write a script **count_files.sh** that takes the following arguments

Path of the directory: This is where you will search for the files
Prefix: Only files whose names begin with the specified prefix shall be counted
Recursive (true or false): This parameter will only take values "true" or "false". If recursive is set to true then you also need to search for the subdirectories in the path mentioned. Otherwise you will only report the files present in the root of the directory mentioned

Note: Script has to give the output to stdout. If the path is invalid (not a directory) or passed parameters are not valid then your script should output relevant error messages and exit with error code -1.

bash ./count_files.sh <directory_path> <prefix> <recursive>

Examples:

Directory structure of /Users/mtech

```
.
├── 1.txt
├── 2.txt
├── abc1.txt
├── abc2.txt
└── subdir
    ├── 4.txt
    └── abc3.txt
```

```
bash ./count_files.sh . abc true
3
```

Recursive search is enabled thus sub directory is included. For the directory path I have passed "." which means current directory. Files counted are abc1.txt, abc2.txt and abc3.txt.

```
bash ./count_files.sh /Users/mtech abc true
3
```

Recursive search is enabled thus sub directory is included. Files counted are abc1.txt, abc2.txt and abc3.txt

```
bash ./count_files.sh /Users/mtech abc false
2
```

Recursive search is disabled thus sub directory is excluded. Files counted are abc1.txt and abc2.txt/

```
bash ./count_files.sh /Users/mtech "" false
6
```

Prefix passed is empty thus the script will count all the txt files.

```
bash ./count_files.sh /Users/mtech/lolo "" false (TODO Check)
/Users/mtech/lolo is not a valid directory
```

and exit with error code -1

```
bash ./count_files.sh . "" true2
recursive should take only true or false values
```

and exit with error code -1

Question 2 (Arithmetic Operations)

Write a script **evaluation.sh** that takes input file as an argument and gives final value after all the arithmetic operations.

You only need to handle +, -, *, / and %. All of these are integer operations. Look at the following examples for better understanding.

Note: Script has to give the output to stdout. If the path is invalid (not a file or invalid file) or passed parameters are not valid then your script should output relevant error messages and exit with error code -1.

Example 1

input.txt

```
10 +
20 -
2 *
4 /
10 +
2 %
```

```
bash evaluation.sh input.txt
1
```

```
0 + 10 = 10
10 - 20 = -10
-10 * 2 = -20
-20 / 4 = -5
-5 + 10 = 5
5 % 2 = 1
```

Example 2

input.txt

```
10 +
20 -
2 *
6 /
10 +
```

```
bash ./evaluation.sh input.txt
7
```

```
0 + 10 = 10
10 - 20 = -10
-10 * 2 = -20
-20 / 6 = -3 (Integer Division)
```

-3 + 10 = 7

Question 3 (Replace text in multiple files in parallel)
Write a Bash script that takes a directory path, a search string, and a replacement string as arguments. The script should find all text files in the specified directory and its subdirectories, replace occurrences of the search string with the replacement string in parallel, and save the changes.

Requirements:

1. The script should accept three arguments: the directory path, the search string, and the replacement string.
2. The script should check if the directory exists and is readable.
3. The script should find all text files in the directory and its subdirectories.
4. The script should use sed to replace occurrences of the search string with the replacement string.
5. The replacements should be done in parallel for better performance.

```
./replace_text_parallel.sh /path/to/directory "old_text" "new_text"
```

Hint: Commands: find, sed, parallel

Question 4 (Disk Hogger)

Create a shell script called diskhog.sh that lists the five largest items (files or folders) in the current directory in decreasing order of size. You should output the sizes in a human readable format like so.

Make sure that your script handles files and directories with spaces in name or names that start with a period.

Check out the man pages for du(1), sort(1), and head(1) (or tail(1)). Read the portion of the bash(1) man page that describes the dotglob option to the shopt bash builtin command. You'll want to use this in your script to make * match files/directories that start with a period.

If the script is run from an empty directory, make sure it prints nothing at all (and not an error message).

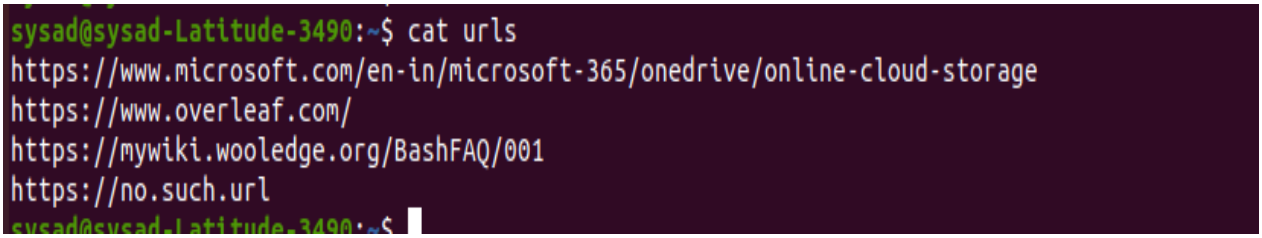
```
$ ./hw1/diskhog
214M    ow
58M     ow.tar.xz
48M     .komodoedit
27M     .cache
2.7M    Desktop
```

Question 5 (URL Testing)

Write a shell script called testurl.sh that accepts a list of URLs in a separate file and tests if each website is up or not. You might find it useful to checkout the curl, wget and tail commands. And check out this link for how to read a file line by line in Bash. It's not obvious.

If any URL in the file isn't accessible, testurl should report that it isn't found as per the example below and should (after testing all URLs), exit with return value 1.

If zero or more than one parameter are passed to testurl, print the usage to stderr and exit with value 1.




```
$ ./testurl urls
Not found: http://no.such.url
$ echo $?
```