COSC 365 Programming Assignment 2

BASH and awk

This assignment is broken into four parts, which are outlined below. To complete these problems, you will be using BASH to compose standard UNIX programs in powerful ways. For every program you might use, I highly encourage you to use the built-in documentation found in the UNIX `man` pages.

For this assignment, you will submit four bash files (one for each part) named as follows:

```
part1.sh  part2.sh  part3.sh  part4.sh
```

You will be given a file called `pa2.tar.gz`, which contains eight directories:

```
part1_ref/  part2_ref/  part3_ref/  part4_ref/
part1_start/  part2_start/  part3_start/  part4_start/
```

Your job for each of these parts will be to write a BASH script that starts with a "start" directory and mimics the provided "ref" directory and each of its files exactly. To run a manual test, you might do the following:

```
UNIX$ cp -r part1_start part1
UNIX$ cp part1.sh part1
UNIX$ cd part1
UNIX$ ./part1.sh
```

A script called `grade.sh` will also be provided that will run your script and check for differences between your new directory and the reference directory. It can be be run this way:

```
UNIX$ ./grade.sh part1
```

This script will create a copy of (for example) `part1_start` named `part1`, copy your script into that directory, `cd` into it, run your script, delete your script (the copy), and finally check for differences in the resulting directory against the reference directory.

NOTE: this script will delete your `part1` directory if it exists before running your script!!! If you want to save your output from a previous test, move it to a new directory name.

General Rules for all Parts

- You may only use standard UNIX utilities already included on the machine.
- You may NOT copy any files/directories from the reference directories.
- Your scripts should not create any files (temporary or otherwise) that are not in the reference directory. The point of this assignment is to take advantage of the UNIX pipe functionality and "filter" programs. Temporary files are not needed.
- You may NOT use any "programming language" tools other than `BASH`, `awk`, and programs like `grep` that use Regular Expressions. For example, `python` and `perl` are not allowed.

Part 1 (20%)

For this part, you will be given a directory with personal record files that look like this: (ex. `aleah.txt`)

```
id 1
age 48
email aleah@yahoo.com
```

Your task is to decompose these record files into directories, where each property is now its own file:

```
aleah/
    id.txt    (contains the text "1")
    age.txt   (contains the text "48")
    email.txt (contains the text "aleah@yahoo.com")
```

General Strategy:

- Use globbing or subprocess substitution to loop over every personal record file.
- Use `basename` or BASH regex expansion to remove the file extension, leaving just the name as a string.
- Use `mkdir` to create a directory for the name.
- Use tools like `grep` and `cut` or `awk` to extract each property from the record.
- Use file redirection to `echo` the property into the appropriate file.
- Remove the old personal record file.

Part 2 (20%)

The objective of part 2 is to process a CSV (comma-separated value) file with car statistics, producing a file called `answer.txt` that contains the make of the car with the highest horsepower and the actual horespower value. For part 2, you may NOT use `awk`.

General Strategy:

- Keep variables for the make of the highest horsepower car and the highest horespower value.
- Loop through each line in the CSV file. You can use code that looks like this to get each line in a variable:

```
while read -r line; do
    # do something with 'line'
done < cars.csv
```

- Be sure to skip the first line (it is the header).
- Extract the make (1st column) from this line by using either `cut` or BASH regex expansion (a good regex for this might be `",*"`).
- Extract the horsepower from this line by using `cut`.
- Compare the extracted horsepower value with the stored max value. If it's larger, set the new max and make. NOTE: the `test` utility (a.k.a `[`) does not support floating point arithmetic or comparisons. In order to compare the horsepower values, take advantage of the `bc` program (basic calculator) by doing something like this:

```
if [ $(echo "$hp > $max_hp" | bc) == 1 ]; then
    ...
fi
```

- After processing all lines, `echo` the variables keeping track of the max make and horespower out to the file `answer.txt`. It should match the formatting in the "ref" file.

Part 3 (20%)

The purpose of part 3 is to demonstrate the usefulness of `awk`. You will be performing the exact same task as part 2, but will use only `awk` to accomplish the task. Your script should look something like this:

```
#!/usr/bin/env bash
awk -F',' '<your awk script>' < cars.csv
```

General Strategy:

- Use `awk`'s line processing, field extraction, variables, and control flow features to complete the same task as part 2.
- You should notice that the `awk` solution is MUCH simpler than part 2.

Part 4 (40%)

The final part of this assignment will involve producing some statistics about DNA sequence data. Unfortunately, the scientists who collected these sequence samples have not combined their samples into a single sequence. Your job will be to extract the sequence samples from each file and put them back together in the correct order so that you can perform some analysis on the whole sequence. The sample files look like this: (ex. `03-12-2023 11:39 AM.sample`)

```
sequence position 17
collected by   alis on 03-12-2023 11:39 AM
sequence data: TTTGTGGACACCTGGCGAAGGTAAAAG...
```

The sequence position indicates where this sample belongs in the whole sequence. It is zero-based, so position 0 is the first sample, and 17 is the 18th sample. The final line in the file contains the actual sequence data that you will be combining with the other samples in order to create the final sequence. The combined sequence should be all in one line in a file called `sequence.txt`.

Once you have the combined sequence, you will summarize the following statistics about the file:

› Total number of each base pair (output to `pairs.txt`).

- Each A in the sequence is an AT pair.
- Each T in the sequence is a TA pair.
- Each C in the sequence is a CG pair.
- Each G in the sequence is a GC pair.

› Distribution of unique codons (output to `codons.txt`).

- A codon is a sequence of three base pairs. For example:

  This sequence snippet: ATCTGA

  has two codons: ATC and TGA

- For each unique codon encountered in the sequence, you will output the percentage of all codons represented by the triplet. For example, if you have 500 total codons and 100 of them are ATA, then you would have the following line in `codons.txt`:

  ```
  ATA 20%
  ```

General Strategy (Creating the Combined Sequence):

- Use `awk` to print only the position followed by the sequence data on one line for each file:

```
4 CCCTGTCGGTACGGTCGCAAGTGTGGGCCTCCAAAATGAGTGAATCG...
8 GCAGAAGTTATACACTTTTTCAATGAACACGCACTCAGAGCTCATAA...
5 TGTTACAATTGCTTTTAGGATTGCGAAATCACACGAGCCTGGGGGGG...
```

- Sort that output stream numerically with `sort` to get each sample in the correct order.
- Use `cut` to remove the sequence position, leaving only the sequence data.
- Remove the newlines using `tr` so that all data is on one line.
- Redirect the final output to `sequence.txt`. You've combined the sequence samples!

General Strategy (Pair Count):

- Use `fold` to split the sequence file such that each character (base pair) is on its own line.
- Have `awk` process the split file and do the following:
  - Use an associative array to count each occurrence of each base pair.
  - At the end of the input, print the four base pairs followed by their count on separate lines in this order: A, T, C, G
  - Redirect the output to `pairs.txt`.

General Strategy (Codon Distribution):

- Use `fold` to split the sequence file such that every three characters (codon) is on its own line.
- Have `awk` process the split file and do the following:
  - Use an associative array to count each occurrence of each codon.
  - Keep a running total of the number of codons.
  - At the end of the input, print each unique codon followed by its distribution in the file as a percentage.
  - Sort the output by codon name.
  - Redirect the output to `codons.txt`.

Grading

- **Program Correctness: 50%**

Does your program run correctly and produce the right output for the given inputs? This part of your grade can be computed by using the provided grading script.

- **Paradigm Adherence: 50%**

Does your program solve the problem using the concepts discussed in class and required from this write up? Your program must use the features of BASH required in each part and focus on the composition of programs as filters to solve the problem.

NOTE: Your program will be graded for correctness by running it with the provided grading script on the Hydra lab machines. Ensure that you test on those machines before submitting so that you may be confident in that portion of your grade.

Tips

- Use the grading script to determine where your output differs from that of the reference directories in order to narrow down missing functionality or output formatting.
- Use your `man` pages!
- The tasks in this assignment seem much harder than they actually are. Let BASH and the UNIX tools do the hard work for you!