School of Electronics and Computer Science Faculty of Physical and Applied Sciences

University of Southampton

COMP1204- Unix Coursework

27634043
Scott Williams

February 25, 2016

1 Scripts

The file, *countreviews.sh*, was created as follows:

```
nano countreviews.sh
```

This command tells the operating system to start a new file called *countreviews.sh* and edit it using a text editor called *nano*.

```
#!/bin/bash

folder="$1"
outputfile=./output
```

The first line, *#!/bin/bash*, tells the OS to run this file as a bash script. That is a script that runs using common command line commands as found in the path to the shell specified in this line.

The second line, *folder="$1"* creates an instance variable called *folder* and saves the contents of the first parameter when running this code to it.

outputfile=./output in the third line creates a new file called output in the same folder as the script.

```
for file in "$folder"/*
do
number=`grep -c "<Author>" "$file"`
filename=`basename $file .dat`
echo "$filename $number" >> $outputfile
done
```

In this section the program loops over every file in the folder and counts the occurrences of the phrase *<Author >* in every file.

The lines *for file in "$folder"/* do ... done* begin and end a for each that loops over the files stored in the instance variable *folder*.

The lines within the for each loop do the data gathering and formatting. The line *number=`grep -c "<Author >" "$file"`* does the actual counting and stores it in an instance variable called *number*. This is done using the *grep -c* command which finds instances of the words in the file.

The line *filename=`basename $file .dat`* finds and formats the file name and stores it in an instance variable called *filename*. This is done using the unique command *basename* which removes the path from the file name and, with the option *.dat* added, it removes the file extension too.

And finally the line *echo "$filename $number" >>$outputfile* stores the two variables together in the *ouputfile*.

```
sort -k2 -n -r $outputfile > sorted.out

cat sorted.out
rm ./output
rm ./sorted.out
```

Finally, in the last section of this code, the output is prepared, printed, and the temporary files removed.

The first line, *sort -k2 -n -r $outputfile >sorted.out* numerically sorts the contents of the *outputfile*, which at this point contains every hotel name and every number of ratings, in reverse order by using second column (that being the column with the data) and stores it in a new temporary file called *sorted.out*.

The next line, *cat sorted.out* outputs all the contents of *sorted.out*, which now contains all the data sorted to our liking. The following lines remove both temporary files thus leaving no trace.

The program can be run, when in the same location as the reviews folder, by using the command:

```
./countreviews.sh reviews_folder/
```

The output is as follows:

```
hotel_218524 2686
hotel_149399 1551
hotel_208454 1223
...
```

2 Hypothesis Testing

2.1 Average Ratings

The file, *averagereviews.sh*, was created as follows:

```
nano averagereviews.sh
```

The program is similar to the previous script which merely counted the number of reviews in each file. However, this bash program finds the average rating for every hotel.

It does this by first finding out how many reviews there are in each hotel's file and then sums the overall rating each reviewer gave to that hotel.

The main bulk of the script is similar to the previous but there are two lines where the magic happens: at the line *rating=$(grep "<Overall >" "$file" — grep -oP '(? <= <Overall >)[0-9]+' — awk 'SUM+= $1 END print SUM')* the sum of all the ratings in the file is found and then stored in the instance variable *rating*, and at the line *average=‘bc -l <<<$rating/$number‘* the average rating is calculated by dividing the sum of the ratings by the total number of reviews in the file.

```
for file in "$folder"/*
do
SUM=0
filename=‘basename $file .dat‘
number=‘grep -c "<Overall>" "$file"‘
```

```
rating=$(grep "<Overall>" "$file" | grep -oP '(?<=<Overall>)[0-9]+' | awk '{SU
average=`bc -l <<< $rating/$number`
echo "$filename $average" >> $outputfile
done
```

The end of the program is similar to that of the script in section one. That is to say that the output is printed out and the temporary files removed as expected.

```
#Sort outputfile
sort -k2 -n -r ./output > ./sorted.out
#Echo outputfile
cat ./sorted.out
#Remove temporary files
rm ./output
rm ./sorted.out
```

The program can be run, when in the same location as the reviews folder, by using the command:

```
./averagereviews.sh reviews_folder/
```

It produces this output:

```
hotel_188937 4.77952755905511811023
hotel_203921 4.77777777777777777777
hotel_230572 4.75000000000000000000
...
```

2.2 Significance

The file *statistical _sig.sh*, was created as follows:

```
nano statistical_sig.sh
```

This program runs a two tailed t statistic test on the highest two rated hotels in the dataset to determine whether there exists a statistically significant difference between them or whether their difference is caused by the randomness of the data.

It does this by first finding the standard deviation which is calculated by:

$$\sqrt{E(X^2) - E(X)^2} \tag{1}$$

The script finds the expected value of the squares through the same way as the previous script by looping through the files and summing the overall rating given by each reviewer. However, as opposed to in the previous script, the overall ratings are squared before summed as shown in this line which is different:

```
rating=$(grep "<Overall>" "$file" | grep -oP '(?<=<Overall>)[0-9]+' | awk '{SU
```

The program can be run, when in the same location as the reviews folder, by using the command:

```
./statistical_sig.sh reviews_folder/
```

3 Discussion

The company behind these files, TripAdvisor, faces many challenges in storing their reviews in such a manner.

One such problem is that data managers in their organisation have to use Bash scripting and commands to access their hotel data. As has been demonstrated, this drains the soul of the programmer and should be avoided.

Additionally, there are issues with the storage of the review data. By storing the reviews for every hotel in individual hotels they prolong the run time by forcing any script that using that data to have to access many hundreds of files. Accessing files is a time expensive endeavour and should be avoided. A better solution would store every review in one large file, distinguishing them perhaps with a new $<Hotel>$ section, thereby decreasing runtime.

Another issue with the data is that the overall score given by the reviewer is completely independent to the scores given in the specific categories in the review (cleanliness, price, etc). A better system might automatically generate a weighted score based on what the reviewer gave in the other categories.

Nevertheless, if it works then TripAdvisor has not real reason to change unless the waiting times for data processing becomes truly unbearable.