COMP1204
Unix Coursework

Alked Ejupi

School of Electronics and Computer Science
University of Southampton
Student ID: 30352118
ae7g18@soton.ac.uk

March, 2019

Contents

1 Scripts

In this section there will be discussed the scripts developed to analyse TripAdvisor's dataset. Before starting to write the script, the dataset was analysed in order to identify the structure and the pattern of each file with extension `.dat` which describes one specific hotel.

1.1 Basic file processing

The first script consists in counting the number of reviews that each hotel has received. The most reasonable way to do that is to count the number of occurrences of `<Author>`. The source code of the script named *countreviews.sh* is as follows:

```
1  #!/bin/bash
2
3  folderReviews="$1"
4  tag="<Author>"
5
6  cd $folderReviews
7
8  grep -c "$tag" * |
9  sed "s/\.dat:/ /g" |
10 sort -k 2 -nr
```

Script 1: countreviews.sh

The first line represents the *shebang*, which tells the interpreter to execute the file by using *Bash Shell*. In the second and third line, there are two variable, `folderReviews` which stores the *positional parameter* assigned by the shell's first argument which is referenced as `$1`, and `tag` which stores the element that we are looking for. Even tough the usage of these variables was not required for the functionality of the script, the only purpose was to make the code more readable and maintainable for future improvements.

The first command of the script is `cd` which allows to go inside the directory, either with an absolute or relative path. By working inside the directory, the script avoids to iterate in each file as the `grep` command will do that for us. By just executing *line 8*, it returns the number of occurrences of `$tag` within each file. This was done by using the option `-c`. By running the `grep` inside the folder, we obtain most of what we are looking for. (see Fig. 1)

```
hotel_100504.dat:125
hotel_100505.dat:163
hotel_100506.dat:33
...
```

Figure 1: Output of `grep -c "$tag" *`

However, the output desired is the name of the hotel and the number of reviews, each separated by a single space, therefore to obtain that, the output is piped to the `sed` command which will remove the substring `.dat:` and replace it with a single space. Lastly, the output obtained is piped to the `sort` command which will sort the number of reviews in descending order. There are 3 options given to it:

- `-k2`: sort the second column.
- `-n`: sort according to string numerical value.
- `-r`: reverse order, since the `sort` command outputs by default in an ascending order.

The final output is described in Fig.2.

```
hotel_218524 2686
hotel_149399 1551
hotel_208454 1223
hotel_150841 1213
...
```

Figure 2: Output of *countreviews.sh*

1.2 Data Analysis

The second script consists in ranking the hotels according to their average overall rating.

```
#!/bin/bash

folderReviews="$1"
tag="<Overall>"

for hotel in ${folderReviews}/*.dat
do
  average=$(
  grep "$tag" $hotel |
  sed "s/$tag//"     |
  awk '{sum+=$1; scores++} END { printf "%.2f", sum/scores}'
  )
  echo "$(basename $hotel .dat) $average"
done | sort -k 2 -nr
```

Script 2: averagereviews.sh

In this script a `for` *loop* was implemented to analyse the average score of each hotel. The *loop* iterates over all files with extension `.dat` which are contained in the folder passed as an argument.
As it can be seen from Script 2, the body of the *loop* runs three commands in parallel, `grep`, `sed` and `awk` respectively, by using the *pipe* "`|`" operator. The `grep` command without any options, will match all the lines containing "`<Overall>`" with their respective score. (see Fig. 3)

```
<Overall>5
<Overall>4
<Overall>3
<Overall>5
<Overall>3
...
```

Figure 3: Output of `grep`

This output is piped to `sed` which will remove the tag `<Overall>` and replace it by an empty character leaving only the scores (Fig. 4).

```
5
4
3
5
3
...
```

Figure 4: Output of `sed`

The last command will be `awk` which contains two steps. First, it will sum all the numbers in the first column (`sum+=$1`) and count the numbers of scores (or *lines*), whereas "`END {printf "%.2f", sum/scores}`" will compute the average and print the output rounded to two decimal places. Finally, the output of the three parallel commands will be stored in `$average` variable and it will be printed out with the name of the hotel just analysed. The final output is sorted in the same way was as *countreviews.sh* (see Fig. 5) .

```
hotel_203921 4.78
hotel_188937 4.78
hotel_230572 4.75
hotel_185406 4.74
hotel_190664 4.73
hotel_188961 4.73
hotel_193121 4.72
...
```

Figure 5: Output of *averagereviews.sh*

2 Discussion

The approach that TripAdvisor uses to collect hotel's reviews leads to several issues when it comes to make further analysis due its unstructured data format. The only way to identify a specific attribute of a file would be iterating over each line which is time-consuming and inefficient. It also involves many steps which can be easily avoided by using a different structure. By using a database instead, the data will be stored and organised in a better way, making data analysis faster and efficient. By just using simply queries the data that we are interested can be easily retrieved, without the need to read all attributes of each file's hotel.

Collecting the reviews from all users and ensure that they are trustworthy is a key aspect for the website. However, the authors of the reviews are not identified as real customer of the hotel. Any user can review any hotel, even though he/she has visited the hotel at all. One solution to ensure that each author is a customer of a hotel is asking for an unique booking reference which will will be checked by the TripAdvisor's system with hotel's database. By adopting this solution, each author will be honest and less probable to be a fake account.

There a few challenges that TripAdvisor faces with its flat file-structure. One of them is duplication of data. Not only it will waste more time to store the information duplicated, but it will also increase the size of it. For example, if an user wrote a review twice accidentally, it would be stored anyway in their file. As a result the review ranking would be altered as it contains two reviews from the same author. Another issue is the difficulty of accessing data. In the case of retrieving a specific information contained in a large file, the algorithm would need to be changed for that specific information. The solution to this, as mentioned previously, it would be using simple queries by implementing a database structure.