COMP1204: Unix Coursework

Huw Jones
27618153

February 8, 2016

1 Scripts

1.1 Count Reviews: Single File

1.1.1 Source

```
#!/bin/bash
grep -c "<Author>" $1
```

1.1.2 Explanation

```
grep -c "<Author>" $1
```

As every review has an author, it made sense to search for the "<Author>" string. Using the "-c" argument for grep returns a search count, rather than the string of occurrences. "$1" is used to get the first (real) argument when the script is executed on the command line.

1.2 Count Reviews: Directory

1.2.1 Source

```
#!/bin/bash

function getReviewCount () {
  grep -c "<Author>" $1
}

# If provided argument is a directory
if [ -d $1 ]
then

  # Loop through files and do reviewcount
  for file in $1/*
  do
    getReviewCount $file
  done;

else
  getReviewCount $1
fi
```

1.2.2 Explanation

```
function getReviewCount () {
  grep -c "<Author>" $1
}
```

This function returns the number of reviews in a hotel data file.

```
if [ -d $1 ]
then

...
fi
```

This checks whether the argument passed was a directory, and if so, execute the appropriate code.

```
for file in $1/*
do
  getReviewCount $file
done;
```

This section is executed if the argument passed as a directory. It iterates over the files in the directory, then calls the "getReviewCount" function for each file.

```
else
  getReviewCount $1
fi
```

This code is executed if the argument passed to the script is not a directory. It executes the "getReviewCount" as it should normally do for a file.

This script works for both 3.1.1: 1 and 2.

1.3 Count Reviews: Sorted

1.3.1 Source

```
#!/bin/bash

function getReviewCount () {
  grep -c "<Author>" $1
}

# If provided argument is a directory
if [ -d $1 ]
then

  reviewCount=""

  # Loop through files and do reviewcount
  for file in $1/*
  do
    # Trim directory prefix and .dat suffix from hotelName to get hotel_xxxx
    hotelName=${file#$1/}
    hotelName=${hotelName%.dat}

    # Set current count to reviewCount:file
    currentCount="$(getReviewCount $file) $hotelName"

    # Append currentCount to reviewCount using a newlinw (\n)
    reviewCount="$reviewCount"$'\n'"$currentCount"
  done;

  # Pipe $reviewCount into a reverse number sort, then format the result using awk
  echo -e "$reviewCount" | sort -nr | awk '{print $2 " " $1}'
else
  getReviewCount $1
fi
```

1.3.2 Explanation

```
reviewCount=""
```

Initialises variable “reviewCount” to an empty string.

```
hotelName=${file#$1/}
```

This removes the directory prefix from “$1” (from when the script was called) from “$file” and assigns it to “hotelName”.

```
hotelName=${hotelName%.dat}
```

Removes the “.dat”” suffix from “hotelName” and assigns it to “hotelName”.

```
currentCount="$(getReviewCount $file) $hotelName"
```

Sets “currentCount” to the output of “getReviewCount” with the hotel name appended to it. This makes it easier to use sort.

```
reviewCount="$reviewCount"$'\n'"$currentCount"
```

Appends “currentCount” to the “reviewCount” using a newline so we can maintain 1 hotel per line.

```
echo -e "$reviewCount" | sort -nr | awk '{print $2 " " $1}'
```

Echoes the contents of “reviewCount” to sdtOut whilst maintaining escape characters. The contents of “reviewCount” is then piped into the stdIn of sort, which reverse sorts the hotel list by number of reviews. Finally, awk prints out the sorted list in the format [hotel] [review count]. The list was originally stored as [review count] [hotel] as this makes it easier to use sort as we don’t have to specify a sort column. Using awk to print out the list as required was child’s play.

1.4 Average Reviews

1.4.1 Source

```
#!/bin/bash

# Gets number of reviews
function getReviewCount () {
  grep -c "<Author>" $1
}

# Gets average score to 1dp
function averageScore() {
  t=$1
  n=$2

  mean=$((t/n))
  dp=$(( 10 * (t % n) / n ))

  echo "$mean.$dp"
}

# Gets the sum of all review scores
function getTotalScore() {
  Scores=$(grep "<Overall>" $1)
  TotalScore=0
  while read -r line; do
    TotalScore=$((TotalScore + ${line:(-1)}))
  done <<< "$Scores"
  echo $TotalScore
}

# Gets the average review of the hotel
function getAverageScore() {
  HotelFile=$1

  ReviewCount=$(getReviewCount $HotelFile)
  TotalScore=$(getTotalScore $HotelFile)
  echo $(averageScore $TotalScore $ReviewCount)
}

# Gets the hotel ID from hotel file
function getTrimmedHotelFile() {
  filePath=$1
  fileName=$2
  hotelName=${fileName#$filePath/}
  hotelName=${hotelName%.dat}
  echo $hotelName
}

# Checks argument passed was a directory
if [ -d $1 ]
then
  # Loops through all files and prints HOTEL_ID AVERAGE_REVIEW
  for file in $1/*
  do
    echo $(getTrimmedHotelFile $1 $file) $(getAverageScore $file)
  done
else
  echo "$1␣is␣not␣a␣valid␣directory."
fi
```

1.4.2 Explanation

```
# Gets average score to 1dp
function averageScore() {
  t=$1
  n=$2

  mean=$((t/n))
  dp=$(( 10 * (t % n) / n ))

  echo "$mean.$dp"
}
```

This function prints the result of $1/$2 to stdOut. As most UNIX shells only deal with integer arithmetic, a method that would produce a mean to 1 decimal place was required. It uses the modulo operator "%" to calulate the remainder, then, by multiplying the remainder by 10 and dividing by the number of reviews, it produces an integer of the first decimal place. This can then be concatenated onto the end of the original, integer, mean.

```
# Gets the sum of all review scores
function getTotalScore() {
  Scores=$(grep "<Overall>" $1)
  TotalScore=0
  while read -r line; do
    TotalScore=$((TotalScore + ${line:(-1)}))
  done <<< "$Scores"
  echo $TotalScore
}
```

This function calculates the total of all the review scores ($\sum x$). It uses grep to filter the lines that contain the overall score (<Overall>). Then, it iterates over each individual score line. As the scores are out of a maximum of 5, the last character of the line is the score. Using bash string formatting, the last character is cut out, and added to the "TotalScore". The total score is then output to stdOut.

```
# Gets the average review of the hotel
function getAverageScore() {
  HotelFile=$1

  ReviewCount=$(getReviewCount $HotelFile)
  TotalScore=$(getTotalScore $HotelFile)
  echo $(averageScore $TotalScore $ReviewCount)
}
```

This function uses the results of two of previous functions (getReviewCount and getTotalScore) and passes them to "averageScore". The result of "averageScore" is then printed to stdOut.

```
# Gets the hotel ID from hotel file
function getTrimmedHotelFile() {
  filePath=$1
  fileName=$2
  hotelName=${fileName#$filePath/}
  hotelName=${hotelName%.dat}
  echo $hotelName
}
```

This function gets the hotel ID in the form of hotel_xxxxxx. It uses string manipulation to remove the directory path from the prefix of the filename Then it removes the .dat file extension. Again, the output of this function is printed to stdOut.

```
# Checks argument passed was a directory
if [ -d $1 ]
then
  # Loops through all files and prints HOTEL_ID AVERAGE_REVIEW
  for file in $1/*
  do
    echo $(getTrimmedHotelFile $1 $file) $(getAverageScore $file)
  done
else
  echo "$1 is not a valid directory."
fi
```

This part of the script checks a directory was passed as the first argument. If a directory wasn't passed as the first argument, the script notifies the user, then terminates. Otherwise, it loops over every file in the data directory, then prints the hotel ID with the average review score.

2 Hypothesis Testing

3 Discussion