COMP1204: Unix Coursework

Huw Jones
27618153

February 13, 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
}

# Gets the hotel ID from hotel file
function getTrimmedHotelFile() {
  echo $1 | sed -e 's:^[/a-zA-Z_0-9\-]*\/::' -e 's:.dat::'
}

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

  reviewCount=""

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

    # Set current count to hotel_id count
    currentCount="$hotelName"$'\t'"$(getReviewCount_$file)"

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

  # Pipe $reviewCount into a reverse number sort
  echo -e "$reviewCount" | sort -k2nr
else
  hotelName=$(getTrimmedHotelFile $1)
  echo $hotelName $(getReviewCount $1)
fi
```

1.3.2 Explanation

```
reviewCount=""
```

Initialises variable “reviewCount” to an empty string.

```
function getTrimmedHotelFile() {
  echo $1 | sed -e 's:^[/a-zA-Z_0-9\-]*\/::' -e 's:.dat::'
}
```

This functions uses sed substitution to remove all directory prefixes up to the last forward slash in the file path (/) from “$file”. It then uses another substitution to remove the .dat extension. Then, it assigns the result to “hotelName”.

The regex works as follows.

```
^
```

Matches from the start of the string

```
^[/a-zA-Z_0-9\-]*
```

Matches 0 or more all alphanumeric characters, dashes, forward slashes and underscores from the start of the string.

```
\/
```

Matches a forward slash. The backslash is used to escape it as forward slash is a special character.

```
^[/a-zA-Z_0-9\-]*\/
```

So, overall this regex matches all directories up to the last slash before the file name. This allows the script to remove the directory prefix. If sed supported “\s\S”, I would have used that instead of the alphanumeric mess that is currently used.

```
currentCount="$hotelName"$'\t'"$(getReviewCount␣$file)"
```

Sets “currentCount” to the hotel ID and the output of “getReviewCount”. The tab is used to make the output prettier than just using spaces.

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

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

```
echo -e "$reviewCount" | sort -k2nr
```

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. Sort takes a “-k” argument that is used to specify the sort column. Here, “-k2nr” sorts the string by the second column (number of reviews)(“k2”), as a numeric type (“n”), reversely (“r”) (to sort by the greater number first).

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>" $1 | sed -e 's:<Overall>::' -e 's:\r::')
  TotalScore=0
  while read -r line; do
    TotalScore=$((TotalScore + line))
  done <<< "$Scores"
  echo $TotalScore
}

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

  ReviewCount=$(getReviewCount $1)
  TotalScore=$(getTotalScore $HotelFile)
  echo $(averageScore $TotalScore $ReviewCount)
}

# Gets the hotel ID from hotel file
function getTrimmedHotelFile() {
  echo $1 | sed -e 's:^[/a-zA-Z_0-9\-]*\/::' -e 's:.dat::'
}

# Checks argument passed was a directory
if [ -d $1 ]
then
  hotels=""
  # Loops through all files and prints HOTEL_ID AVERAGE_REVIEW
  for file in $1/*
  do
    currentHotel="$(getTrimmedHotelFile $file)"$'\t'"$(getAverageScore $file)"
    hotels="$hotels"$'\n'"$currentHotel"
  done

  # Sort hotels by second column (rating)
  echo -e "$hotels" | sort -k2nr
else
  echo "$(getTrimmedHotelFile $1) $(getAverageScore $1)"
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>" $1 | sed -e 's:<Overall>::' -e 's:\r::')
  TotalScore=0
  while read -r line; do
    TotalScore=$((TotalScore + line))
  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, pipes the grep output to a sed substitution that removes "<Overall>". Next, it adds the line (which should be the rating) 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.

```
# Checks argument passed was a directory
if [ -d $1 ]
then
  # Loops through all files and prints HOTEL_ID AVERAGE_REVIEW
  for file in $1/*
  do
    currentHotel="$(getTrimmedHotelFile $file)"$'\t'"$(getAverageScore $file)"
    hotels="$hotels"$'\n'"$currentHotel"
  done

  # Sort hotels by second column (rating)
  echo -e "$hotels" | sort -k2nr
else
  echo "$(getTrimmedHotelFile $1) $(getAverageScore $1)"
fi
```

This part of the script checks a directory was passed as the first argument. Otherwise, it loops over every file in the data directory, then prints the hotel ID with the average review score, sorted by average review.

2 Hypothesis Testing

3 Discussion