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 reviewcount () {
  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
    reviewcount $file
  done;

else
  reviewcount $1
fi
```

1.2.2 Explanation

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

I turned the count reviews into a function to use later.

```
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
  reviewcount $file
done;
```

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

```
else
  reviewcount $1
fi
```

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

This script works for both 3.1.1: 1 and 2.

1.3 Count Reviews: Sorted

```
#!/bin/bash

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

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

  reviewCount=""

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

    # Set current count to reviewCount:file
    currentCount="$(reviewcount $file) $fileName"

    # 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
  reviewcount $1
fi
```

1.3.1 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="$(reviewcount $file) $hotelName"
```

Sets “currentCount” to the output of “reviewCount” 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.

2 Hypothesis Testing

3 Discussion