Total Points: 100

COP4342 – UNIX Tools

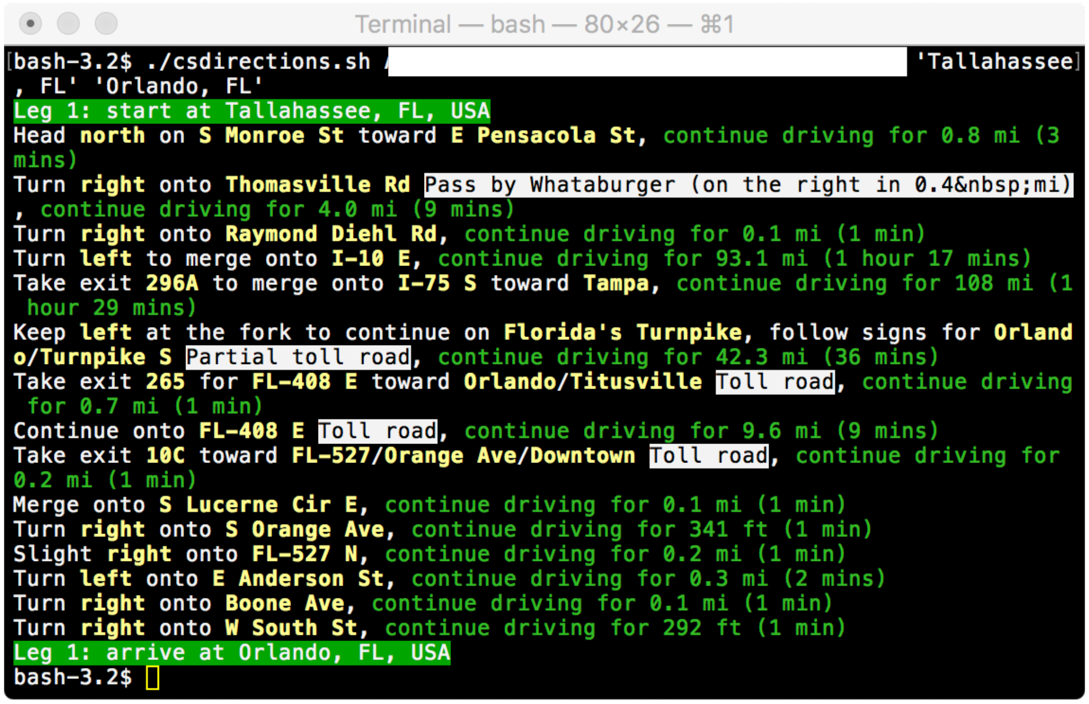
Makeup Assignment: Google Maps Directions API

Instructor: Dr. Robert Van Engelen

Spring 2018

Description: A bash script that produces directions from point A to point B using the Google Maps Directions API.

Assignment: In this programming assignment you will implement a script csdirections.sh that takes an origin address, a destination address and the mode of transportation to produce an optimal route from the origin to the destination. See an example in the figure below:



```
[bash-3.2$ ./csdirections.sh [redacted] 'Tallahassee]
, FL' 'Orlando, FL'
Leg 1: start at Tallahassee, FL, USA
Head north on S Monroe St toward E Pensacola St, continue driving for 0.8 mi (3
mins)
Turn right onto Thomasville Rd Pass by Whataburger (on the right in 0.4&nbsp;mi)
, continue driving for 4.0 mi (9 mins)
Turn right onto Raymond Diehl Rd, continue driving for 0.1 mi (1 min)
Turn left to merge onto I-10 E, continue driving for 93.1 mi (1 hour 17 mins)
Take exit 296A to merge onto I-75 S toward Tampa, continue driving for 108 mi (1
 hour 29 mins)
Keep left at the fork to continue on Florida's Turnpike, follow signs for Orland
o/Turnpike S Partial toll road, continue driving for 42.3 mi (36 mins)
Take exit 265 for FL-408 E toward Orlando/Titusville Toll road, continue driving
 for 0.7 mi (1 min)
Continue onto FL-408 E Toll road, continue driving for 9.6 mi (9 mins)
Take exit 10C toward FL-527/Orange Ave/Downtown Toll road, continue driving for
0.2 mi (1 min)
Merge onto S Lucerne Cir E, continue driving for 0.1 mi (1 min)
Turn right onto S Orange Ave, continue driving for 341 ft (1 min)
Slight right onto FL-527 N, continue driving for 0.2 mi (1 min)
Turn left onto E Anderson St, continue driving for 0.3 mi (2 mins)
Turn right onto Boone Ave, continue driving for 0.1 mi (1 min)
Turn right onto W South St, continue driving for 292 ft (1 min)
Leg 1: arrive at Orlando, FL, USA
bash-3.2$ 
```

Command line options: csdirections.sh script takes four command-line arguments, no options. The first argument is the Google Maps Directions API key (which is private, hence redacted in the figure above), the second argument is the origin address, the third argument is the destination address, and the fourth optional argument is the mode of transportation, which is one of "driving", "walking", "cycling" or "transit" (for public transit), when omitted the transportation mode defaults to "driving".

Command output: the csdirections.sh script should display on stdout the route from the origin to the destination for each leg of the route starting with the output "**Leg <num>: start at <start_address>**" followed by the step-by-step instructions with distance and duration of the form "**<instruction>, continue <travel_mode> for <distance> (<duration>)**" and at the end of the leg should show the output "**Leg <num>: arriving at <end_address>**". The **<start_address>**, **<end_address>**, **<instruction>**, **<travel_mode>**, **<distance>** and **<duration>** are all extracted from the XML response returned by the Google Maps Directions API. More details will follow below.

Exit code: csdirections.sh should always exit with zero, even when no route could be found.

Google Maps Directions API key: visit https://developers.google.com/maps/documentation/directions/get-api-key to obtain a key. You can use this key to query the Google Maps Directions API with your script by executing an HTTP GET on **https://maps.googleapis.com/maps/api/directions/xml** to obtain XML responses with upt-to-date and live directions. We will use the REST API XML responses instead of JSON because XML is much easier to parse in bash scripts by matching XML begin/end tags as strings with **case** pattern or **if [[ ]]; then** patterns in your script. We will also use the **xmllint** utility to reformat XML to remove indentation and to ensure that XML is always in the same line-by-line format before we "parse" it. The REST API expects an HTTP GET with an URL of the form:

```
<URL>?units=imperial&key=<your-API-key>&origin=<origin>&destination=<destination>&mode=<mode>
```

where **<URL>** is the address shown above, **<your-API-key>** is the value of the key you received when registering with the Google Maps Directions API, **<origin>** is a URL-encoded origin address, **<destination>** is a URL-encoded destination address, and **<mode>** is the transport mode. A URL-encoded string uses "percent encoding" of reserved characters, see https://en.wikipedia.org/wiki/Percent-encoding. If you cannot obtain a key for some reason, you can still complete this assignment by implementing the script to read the XML directions from stdin. However, getting a key is recommended and a lot more fun to use to get real-time directions!

Using wget: Use **wget** to invoke the Google Maps Directions API as follows:

```
wget -q -O - "$URL?units=imperial&key=$KEY&origin=$FROM&destination=$TO&mode=$MODE"
```

where **URL=https://maps.googleapis.com/maps/api/directions/xml** and **$KEY**, **$FROM**, **$TO**, **$MODE** are provided as command-line arguments, with **$MODE** defaulting to "**driving**" when not provided. Note that option **-O -** (dash after **-O**) sends the response to stdout.

Example XML responses: Two example XML direction responses from the Google Maps Directions API can be downloaded from http://www.cs.fsu.edu/~engelen/courses/COP4342/csdirections.xml (for the example shown in the figure) and from http://www.cs.fsu.edu/~engelen/courses/COP4342/csdirections2.xml to test your script.

Testing your script: To test your script you may not want to send a lot of API requests to Google for each time you run your script to just test your parsing and output. There is a limit on how often you can query the API. Instead, you can save one or two XML responses to files and let your script read from the file e.g. using **cat** instead of using **wget**. Then use **wget** as shown above when you are done testing and ready to submit the assignment.

Script submission: If for some reason you cannot get a Google Maps Directions API key then your script should read XML from stdin so we can test it without deducting points. You should however still process the command-line arguments to produce a valid URL even when you do not use **wget**. On the other hand, if you create a more realistic script using **wget** then simply submit your script as is. To get the response from the Google Maps Directions API. We have our own API key to test your submission. **NEVER include your key in your submission, your key is for PRIVATE use only!** The Google Maps APIs are subject to policies and quotas so you should not share your key with the script for this assignment, see https://developers.google.com/maps/documentation/directions/policies for details.

Helpful hints: to implement csdirections.sh you will likely need the following:

- **wget -q -O - <URL>** to invoke the REST API with HTTP GET URL formatted as shown earlier, sending the XML response to stdout where you will have to pipe it to **xmllint**
- **xmllint --format** to reformat the XML by removing all indentation, which requires the environment variable **XMLLINT_INDENT** to be set to the empty string before invoking **xmllint**. While the Google Maps Directions API XML response is nicely formatted, it is a good idea to reformat XML to a predictable structure to parse it by matching it line-by-line looking for XML opening and closing tags.
- The result of **xmllint** should be piped to a script function that you need to write, which reads the input line-by-line until EOF.
- **while read line; do … done** to repeatedly read a line from stdin until EOF into the variable **line**.
- **case** or **if-then** to match XML tags stored in variable **line**, so you can look for XML opening tags and closing tags that contain the information you are looking for.
- An array to collect steps of each leg, where each step added to the array is formatted as "**<instruction>, continue <travel_mode> for <distance> (<duration>)**" and eventually the array is displayed for the entire leg with **for step in "${array[@]}"; do** as discussed in the lecture notes and in class. You will need other variables to collect the XML information within **<step>…</step> tags** and then assign it to an array entry.
- **tr** to convert characters.
- **sed -e 's/<[^>]*>//g'** to remove XML tags and **-e 's/&amp;/\&/g'** to replace XML **&amp;** entity by a **&**. You may want to translate the entities &apos; &quot; as well. You will need several substitutions, so we use option **-e**.
- **var=$(command)** to get the results of a command assigned to **var**. For example, the following converts **<tag>HELLO</tag>** to **hello**: **var=$(echo "<tag>HELLO</tag>" | tr A-Z a-z | sed -e 's/<[^>]*>//g')**
- **ANSI color codes** are enabled in textual output as **CTRL-ESC [** two-byte sequence, followed by a code and the character **m**. Remember how to enter CTRL-ESC with CTRL-V CTRL-ESC in vi into a string, or you could use **\x1b** (hex) or **\033** (octal) with **echo** or **printf**. Here is a list of some of the ANSI color codes:

1	bold
7	invert video
27	cancel invert video
42	green background
0	normal: reset all colors

  it is fine if you want to use other colors, as long as the colors are visible on a black terminal background and the text output is formatted as shown in the example. For more details on ANSI codes, see https://en.wikipedia.org/wiki/ANSI_escape_code#SGR_(Select_Graphic_Rendition)_parameters. Again, these ANSI color codes are enabled with **CTRL-ESC [** (the two-byte CSI sequence) and ending with **m**. You can use **sed** to substitute text between **<b> … </b>** with **CTRL-ESC [1m … CTRL-ESC [0m** to show text in bold, e.g. using a group capture and a backreference in **s/regex/text/g**.
- Some code to encode origin and destination values as per https://en.wikipedia.org/wiki/Percent-encoding as is required to produce a valid REST API HTTP GET request URL with **wget**. Even if your code does not use **wget** because you have no key to test your script with, you should create this URL to show it is possible to use **wget**.

Submission Requirements: A tar file named ‘yourCSusername_makeup.tar’ containing the following files:-

[1]. The ./csdirections.sh script source code.

Grading Policy: Points distribution:-

[1]. Code Readability	(20 points)
[2]. Test Cases	(80 points)

Individual parts of the implementation will not be graded separately for correctness. There will be several cases to test the implementation logic as a whole. Also, keep in mind that your code will be tested on *linprog*. Students should test their code thoroughly on the *linprog* server before submission

Miscellaneous:

Honor code policy will be strictly enforced. Write the code by yourself and protect your submission.

Key Concepts: *bash programming, Web services REST API, XML, sed scripting*

Shell commands: *case, if, for, while, echo, printf, exit, function, read, wget, xmllint, tr, sed, variables and arrays.*