Module 1: Bash Project – Stock Watcher

Project Brief

In this project you will develop a bash script to download and process historic stock price data.

Successful completion will demonstrate competence in basic bash scripting and awk programming. These skills are valuable in automating and simplifying manifold text and data processing tasks in scientific computing.

The terminal goal is to write an interactive bash script to repeatedly ask the user for a stock ticker symbol (e.g. AAPL - Apple Inc., DOW - Dow Chemical) and a historic month and year (e.g. 9/2008) to produce files containing the adjusted daily closing stock price for each trading day that month.

The desired format of the file is two columns titled "Date" and "Adjusted Closing Price / " listing the closing stock price in US cents for each trading day in that month in reverse chronological order. The file should be titled `<stockSymbol>_<Month>_<Year>.txt`.

Deliverables

You should submit your script via email to by **11:59pm on 30 January 2020.** ***Scripts must run on the EWS Linux machines.*** Late submissions will not be accepted.**

Points will be awarded for the specific expectations listed below, and for the overall script *usability, performance, clarity, and presence of useful in-script comments/documentation.*

It is **strongly advised** that students write the script using vi/emacs or a simple text editor on a machine running Linux or Mac OS X or via ssh to EWS). Scripts written on Windows machines often contain extraneous characters that make them non-portable to other environments.

Specific Expectations

1. [10 pts] Use the read function to obtain and store user input (stock name, month, year) into appropriately named variables.

2. [10 pts] Yahoo! Finance maintains an archive of historic stock data that can be easily accessed using the script available at this URL:

github.com/bradlucas/get-yahoo-quotes For example, to download all historical stock prices for Apple (AAPL) we would execute `./get-yahoo-quotes.sh AAPL`

This will produce a CSV file with all of the data. You will need to modify this script (make a copy of the script, and start editing your own version) to make sure that your script sets start and end dates based on user input. Make sure to automatically use the correct end day depending on the user-selected month (otherwise when you use `date` to convert it to seconds for `START_DATE` and `END_DATE` you will have a problem). We must automatically specify the end day depending on the user-selected month. Special care is necessary for the month of February in leap years.

- *Hint 1:* A case conditional structure predicated on the user-selected month may be useful. Think about nesting a second case conditional block to handle February.
- *Hint 2:* Pseudo-code to determine leap years: en.wikipedia.org/wiki/Leap_year#Algorithm.
- *Hint 3:* The special bash variable `$?` holds the return code for the most recent call made in your script. It will hold a value of 0 if the call was successful, and will hold a non-zero value if the call failed. You may wish to use this special symbol in an if loop (e.g., `if [ $? -ne 0 ] ; then`) to determine whether or not your call to wget succeeded.

3. [20 pts] The raw data downloaded (into `AAPL_2009.txt`, in the above example) will contain a number of fields and a header line containing field descriptors. **Using a call to awk within your script**, convert this file into the desired file format. You will use awk to extract the pertinent fields from the raw data, and generate a file in the desired format, and with the desired title.

- *Hint 1:* By default, awk expects fields to be delimited by white space. We can change the delimiter by calling awk as `awk -F '<delimiter>'`, where `<delimiter>`=comma, semi-colon, colon, etc.
- *Hint 2:* Yahoo finance reports that adjusted closing price in dollars. You will need to have awk do some simple arithmetic within the printf statement to make the $ to conversion.
- *Hint 3:* You will need to use the special awk variable `NR` to ignore header lines in your input file.

- *Hint 4:* When placing multiple variables in a text string, you may need to indicate to bash where one variable ends and the next begins using empty double quotes. For example, if I want to write "Hello, world!" to a file `Doe_John.txt`, and I have variable `firstName=John` and `lastName=Doe`, I would write: `echo "Hello, world!" > "${lastName}_${firstName}`. If we omit the curly braces, bash gets confused because it thinks you are trying to dereference the (nonexistent) variable lastName_.

4. [5 pts] Give some consideration to what should happen if your script encounters an error (i.e., defensive programming). Specifically, what should happen if:

- A non-historic (i.e., future) month is entered?
- The stock ticker symbol is invalid?
- Data is unavailable for the requested month (e.g., stock didn't exist at that time)?
- The user enters January / Jan / 1 to indicate the month?

Perhaps you wish your script to simply produce a blank file, perhaps exit gracefully and tell the user the file is blank, or perhaps give the user another chance to enter a legal symbol/date pair. **Justify your decision**.

5. [5 pts] Make your script to run forever, repeatedly asking the user for symbol/date pairs. You will need to make use of an infinite loop, or recursion (in which the script contains a function that repeatedly calls itself *ad infinitum*). Of course, Ctrl+C will break us out.