Problem Set 1: STAT243

Cristobal Pais - cpaismz@berkeley.edu

September 8th, 2017

Solutions

Problem 2

a) Download, extract and analyze

The general idea of the code consists of downloading the target file, perform any operation needed in order to get access to its content and then use a series of UNIX commands that allow us to extract, process, record, and output some of the elements inside the original file. After downloading the file with the right extension, we rename it for convenience and we extract the regions and countries data using the *grep* command. This can be done due to the presence of the character '+' as a mark for the regions, allowing us to select the lines with and without this special character, and then redirect them to other files.

Once we separated the regions and countries data, we are able to use commands like *grep* in order to find the relevant entries and information from the processed files. At this point, we can obtain the names of the top five countries in terms of Area Harvested for the selected years by using a chain (via piping) of *grep* commands, sorting the data by the value of the relevant parameter (Area Harvested) and print the top five countries ranking using the classic *head* function output as input for *awk*, that allows us to isolate and print the name of the country (same results can be obtained using *cut* command).

Finally, the automatization of the code relies on a simple *for* loop among the relevant years, while invoking the same chain of commands previously used, with a slight modification in order to take into account the current year value denoted by the variable n.

In order to implement our strategy, we perform the following steps:

1. We download the data from the given url taking into account the extension of the file. After downloading it from an external browser, we realized that the extension of the file is a .zip. Hence, we can use the *wget* command with the flag -O in order to output the content of the url into a file, for which we can specify the extension as .zip.

2. After unzipping the file (and removing it from the folder), we rename it using the *mv* command since the original name is too long and not very explicit. We are using the fact that the extracted file is the newest file (last one created) inside the current directory.

```
# Download the file in .zip format
wget -q -O apricots.zip "http://data.un.org/Handlers/DownloadHandler.ashx?"\
"DataFilter=itemCode:526&DataMartId=FAO&Format=csv&c=2,3,4,5,6,7&s=countryName:"\
"asc,elementCode:asc,year:desc"

# Unzip and delete the downloaded file
unzip -q apricots.zip
rm apricots.zip

# Change original name based on last file created
mv $(ls -rt | tail -n 1) apricots.csv
```

3. Based on the text characters, we notice that region names are associated with the special character '+', allowing us to use *grep* with and without the -v flag for creating both files. Headers and footnotes are included/preserved in both files.

```
# Generate regions file and keep the headers
head -n 1 apricots.csv > apricots_regions.csv

# Add regions file content
grep "+" apricots.csv >> apricots_regions.csv

# Adding footnotes
tail -n 7 apricots.csv >> apricots_regions.csv

# Generating countries file, footnotes included
grep -v "+" apricots.csv > apricots_countries.csv
```

4. Once we have both files, we can create a chain expression using pipes between a series of commands in order to filter the countries data set for displaying the top 5 countries in terms of Area Harvested for the specific year (2005). We start filtering by the year and then by the Area Harvested field using *grep*. Then, we sort the output based on the numerical value of this field, using the double quotes as separators (that's why the flag –key uses 12 as its input). Finally, we take the first 5 rows of the file using *head* and we print the name of the country calling *awk* with commas as delimiters.

```
# Filter by year and Area Harvested, then sort and show the top 5 (only names)
echo "Year 2005 Area Harvested ranking"
grep "\"2005\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'
```

5. For checking purposes, we can replicate the same analysis for all the relevant years under study. In order to perform this operation, we just need to copy the previous code (last line) and change the string associated with the year in the first filtering step.

```
# Individual codes: for each year
echo "Year 1965 Area Harvested ranking"
grep "\"1965\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

echo "Year 1975 Area Harvested ranking"
grep "\"1975\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

echo "Year 1985 Area Harvested ranking"
grep "\"1985\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

echo "Year 1995 Area Harvested ranking"
grep "\"1995\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

echo "Year 2005 Area Harvested ranking"
grep "\"2005\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'
```

6. Based on the previous code, we can easily automate the Area Harvested analysis and examine the top five countries for 1965, 1975, 1985, 1995, and 2005 by using a simple for loop among these years. We initialize an auxiliary variable $n = 1965$ that will be updated by ten units per iteration until the value 2005. Inside the loop, we will print some information regarding the year to the main console/screen and then we will have a slight variation of the previous code, taking into account the fact that the year is a variable instead of a string. Thus, the following code will output the top five ranking for all the years under study.

```
# Automated code for years 1965,...,2005 by 10
# For inside the specific set
for n in {1965..2005..10}
do
  # Print the year under study
  echo "Year "${n}" Area Harvested ranking"

  # Print top five Area harvested ranking (structure of previous code)
  grep "\"${n}\"" apricots_countries.csv | grep "Area Harvested" | \
  sort --field-separator='"' --key=12 -nr   | head -n 5 | awk -F "," '{print $1}'
done
```

Based on the output obtained from the previous code, we can conclude that the rankings have changed since 1965 (up to 2005). Turkey is the only country that is present in all the rankings, both in first and second places. Tunisia and Spain appear in 4 out of 5 rankings, loosing their top five status in 2005. Interesting is the presence of the USSR from 1965 (first place) to 1985 (second place below Turkey). The detailed results are as follows:

i) Ranking 1965
   1. USSR
   2. Turkey
   3. United States of America
   4. Spain
   5. Tunisia

ii) Ranking 1975
   1. USSR
   2. Turkey
   3. Spain
   4. Tunisia
   5. Italy

iii) Ranking 1985
   1. Turkey
   2. USSR
   3. Spain
   4. Iran
   5. Tunisia

iv) Ranking 1995

1. Turkey
2. Iran
3. Spain
4. Ukraine
5. Tunisia

v) Ranking 2005

1. Turkey
2. Iran
3. Pakistan
4. Uzbekistan
5. Algeria

b) Bash Function

The main strategy for coding the function consists of creating a .sh file called **Faofunction.sh** where we will define our function, allowing us to call it after invoking the .sh file from the command line, avoiding the declaration of the function within the console. Inside this file, we start by defining a regular expression that will allow us to test if the input given by the user is a valid integer or not. Then, we detect all possible cases using conditional clauses (*if*) to check if the input given by the user is not valid: non-numeric and/or the presence of more/less than one value. Using the same approach, we include a help message using the -h or –help flags, indicating the usage of the function and its description.

Finally, if the input is valid, the function is invoked. It will download the file corresponding to the ID used as input as a zip file. Then it will be unzipped and its content will be printed to the console.

We developed the following code in order to perform the described actions:

1. In a .sh file, we initialize a function called *print_item_data* that takes as main (and only) input a numerical value defined by a regular expression *re*. In order to provide some guidance and information to the user about our function, we include a validation step (first) that looks for the flag -h or –help. If one of these flags is given as input, a helping message will be printed out to the console, indicating the usage of the function and some relevant information about its utility.

```
#!/bin/sh
# Function definition
print_item_data () {
# A regular expression is defined for checking input
re='^[0-9]+$'

# If flag --help or -h is given, output help msg
if [ "$1" == "--help" ] || [ "$1" == "-h" ]; then
    echo "Usage: print_item_data [INTEGER]"
    echo "Downloads and prints the data on agricultural production for item"
    echo "with ID INTEGER"
    echo "Information Provided by United Nations Food and Agriculture Organization (FAO)"
```

2. The second check-in of the input consists of comparing the given input value *$1* with the declared regular expression: if the input is not numeric, an error message will be displayed.

```
# If the input is not a number, output error msg 1
elif ! [[ $1 =~ $re ]]; then
    # Print error msg 1
    echo "error: Input should be only an integer value"
    echo "use print_item_data -h or --help for more information"
```

3. The third validation will check the number of inputs given by the user. If they are less or more than one, an error message will be printed out.

```
# If we have more than 1 input, output error msg 2
elif [ $# != "1" ]; then
    # Print error msg 2
    echo "error: Function needs only one integer as input"
    echo "use print_item_data -h or --help for more information"
```

4.1 If all validations are passed (user gives a valid input), the file associated with the given product ID is downloaded as a .zip.

```
# Else, run the function
else
    # Download the file in .zip format
    wget -q -O FAO_${1}_data.zip "http://data.un.org/Handlers/DownloadHandler.ashx?"\
"DataFilter=itemCode:"${1}"&DataMartId=FAO&Format=csv&c=2,3,4,5,6,7&s=countryName:"\
"asc,elementCode:asc,year:desc"
```

4.2 Note that we can also use auxiliary variables like in the following code in order to break the long url string without changing the identation of the code.

```
# Else, run the function
else
    # Download the file in .zip format
    fileurl="http://data.un.org/Handlers/DownloadHandler.ashx?DataFilter=itemCode:"
    fileurl+=${1}"&DataMartId=FAO&Format=csv&c=2,3,4,5,6,7&s=countryName:"
    fileurl+="asc,elementCode:asc,year:desc"
    wget -q -O FAO_${1}_data.zip ${fileurl}
```

5. Finally, the file is unzipped obtaining a .csv file and its name is changed following a specific and explicit structure. The .zip file is eliminated from the current directory and the content of the relevant text file is printed out to the screen.

```
    # Unzip the downloaded file
    unzip -q FAO_${1}_data.zip

    # Change original name based on last file created
    mv $(ls -rt | tail -n 1) FAO_${1}_data.csv

    # Remove the zip file (optional)
    rm FAO_${1}_data.zip

    # Prints out to the screen
    cat FAO_${1}_data.csv
fi
}
```

6. Thus, we can easily use the previous function (saved in an shell file called Faofunction.sh) and then pipe its output to any other commands (like *less*) using the following code:

```
# Load the function Faofunction.sh
. /Faofunction.sh

# If needed, use the help flag for information
print_item_data -h

# Print out the apricots data (ID=526)
print_item_data 526 | less
```

Problem 3

Download files by extension from url

The main approach for downloading the .txt files contained in the given url consists of downloading the index.html file of the website and then use regular expressions with the command *grep* for parsing, obtaining and save the names of the text files hosted on the relevant directory into an auxiliary text file. Some characters will be modified/replaced in order to match exactly the name of the files (no extra characters will be kept). In addition, all web interaction is performed without printing any output to the screen, not disturbing/confusing the user thanks to the quiet (-q) flag used with *wget*.

Then, the downloading process will begin, showing progress information to the user. The implementation is based on a simple *while* clause that reads every line (*.txt names) inside the auxiliary file previously created and uses them as inputs for downloading all of them from the specific url using the command *wget*.

The code implemented is the following:

1. We start our script downloading the index.html file located in the website directory. This can be accomplished by simply using *wget* command, including the flag -q for not printing out irrelevant information to the user. After downloading it, we obtain, extract and save to an auxiliary file the names of the relevant text files (.txt extension) based on a regular expression and a post-processing step where an extra character ('>') is removed using the *tr* command. This can be done due to the specific structure of the .html file.

   A message indicating that the download process will begin is printed out to the screen.

```
#!/bin/sh
# Get index.html file without output
wget -q https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/

# Get the name of the .txt files and save them to a text file
egrep -oe '>[^<].*\.txt' index.html | tr -d ">" > TextList.txt

# Output current status
echo "Download starts"
```

2. In order to download all .txt files inside the web directory while showing some progress information to the user, we use a *while* loop that allows us to read the auxiliary file line by line (name by name). Thus, we redirect each text file name to a variable *filename* that is used inside the loop for printing out information regarding the status of the file (starting to download it or already downloaded) and for concatenating it with the web directory address in order to directly download each file into the current local working directory using the *wget* command.

   Finally, we remove the index.html file and the auxiliary text file with all the names of the downloaded .txt file from the local machine, for cleaning purposes.

```
# Read txt with names line by line and save it to filename var
while IFS= read -r filename
do
    # Indicates the user the file being downloaded
    echo "File ${filename} is being downloaded"

    # Download the file
    wget -q "https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/"${filename}
    # Tells the user that it has been downloaded
    echo "File ${filename} has been downloaded"

# Declare the input file for reading on while loop
done < TextList.txt

# Remove the index.html and TextList files
rm index.html TextList.txt
```

Problem 4 code

This is the code used for generating the next page of this report. We start with the introduction, including the two r-inline functions for generating the years:

```
The height of the water level in Lake Huron fluctuates over time. Here I 'analyze' the
variation using R. I show a histogram of the lake levels for the period `r 1876 - 1.` to
`r 1000 + 72 + 900`
```

After the introduction, we generate the histogram taking into account its width and height (using fig.width and fig.height options), such that the plot looks good and does not cover more than half of the width:

~~~
```{r, engine='R', fig.width=3, fig.height=4}
hist(LakeHuron)
```
~~~

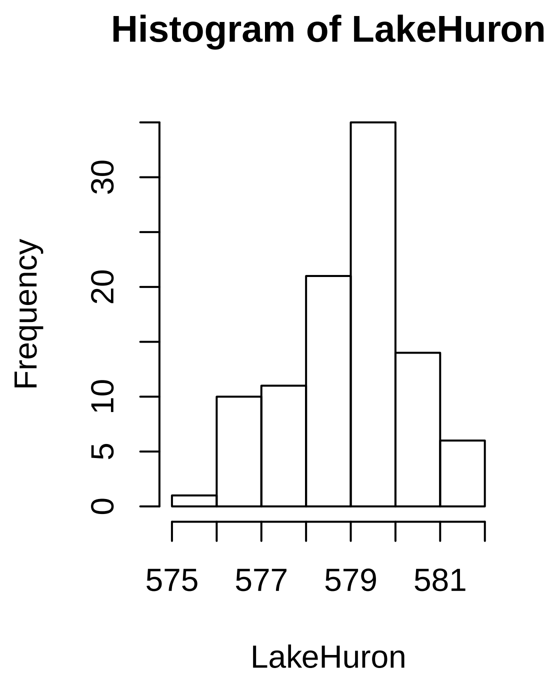
Finally, we define lowHi and yearExtreme variables:

~~~
```{r, engine='R'}
lowHi <- c(which.min(LakeHuron), which.max(LakeHuron))
yearExtrema <- attributes(LakeHuron)$tsp[1]-1 + lowHi
```
~~~

Problem 4: output

The height of the water level in Lake Huron fluctuates over time. Here I 'analyze' the variation using R. I show a histogram of the lake levels for the period 1875 to 1972

```r
hist(LakeHuron)
```



```r
lowHi <- c(which.min(LakeHuron), which.max(LakeHuron))
yearExtrema <- attributes(LakeHuron)$tsp[1]-1 + lowHi
```

Appendix

In this section we include the codes used in this report, separated by questions.

Problem 2

a) Download, extract and analyze

```
# Download the file in .zip format
wget -q -O apricots.zip "http://data.un.org/Handlers/DownloadHandler.ashx?"\
"DataFilter=itemCode:526&DataMartId=FAO&Format=csv&c=2,3,4,5,6,7&s=countryName:"\
"asc,elementCode:asc,year:desc"

# Unzip and delete the downloaded file
unzip -q apricots.zip
rm apricots.zip

# Change original name based on last file created
mv $(ls -rt | tail -n 1) apricots.csv

# Generate regions file and keep the headers
head -n 1 apricots.csv > apricots_regions.csv

# Add regions file content
grep "+" apricots.csv >> apricots_regions.csv

# Adding footnotes
tail -n 7 apricots.csv >> apricots_regions.csv

# Generating countries file, footnotes included
grep -v "+" apricots.csv > apricots_countries.csv

# Filter by year and Area Harvested, then sort and show the top 5 (only names)
echo "Year 2005 Area Harvested ranking"
grep "\"2005\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

# Individual codes: for each year
echo "Year 1965 Area Harvested ranking"
grep "\"1965\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

echo "Year 1975 Area Harvested ranking"
grep "\"1975\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

echo "Year 1985 Area Harvested ranking"
grep "\"1985\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

echo "Year 1995 Area Harvested ranking"
grep "\"1995\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

echo "Year 2005 Area Harvested ranking"
grep "\"2005\"" apricots_countries.csv | grep "Area Harvested" | \
sort --field-separator='"' --key=12 -nr | head -n 5 | awk -F "," '{print $1}'

# Automated code for years 1965,...,2005 by 10
# For inside the specific set
for n in {1965..2005..10}
do
  # Print the year under study
  echo "Year "${n}" Area Harvested ranking"

  # Print top five Area harvested ranking (structure of previous code)
  grep "\"${n}\"" apricots_countries.csv | grep "Area Harvested" | \
  sort --field-separator='"' --key=12 -nr   | head -n 5 | awk -F "," '{print $1}'
done
```

b) Bash Function

```
#!/bin/sh
# Function definition
print_item_data () {
# A regular expression is defined for checking input
re='^[0-9]+$'

# If flag --help or -h is given, output help msg
if [ "$1" == "--help" ] || [ "$1" == "-h" ]; then
    echo "Usage: print_item_data [INTEGER]"
    echo "Downloads and prints the data on agricultural production for item"
    echo "with ID INTEGER"
    echo "Information Provided by United Nations Food and Agriculture Organization (FAO)"

# If the input is not a number, output error msg 1
elif ! [[ $1 =~ $re ]]; then
    # Print error msg 1
    echo "error: Input should be only an integer value"
    echo "use print_item_data -h or --help for more information"

# If we have more than 1 input, output error msg 2
elif [ $# != "1" ]; then
    # Print error msg 2
    echo "error: Function needs only one integer as input"
    echo "use print_item_data -h or --help for more information"

# Else, run the function
else
    # Download the file in .zip format
    wget -q -O FAO_${1}_data.zip "http://data.un.org/Handlers/DownloadHandler.ashx?"\
"DataFilter=itemCode:"${1}"&DataMartId=FAO&Format=csv&c=2,3,4,5,6,7&s=countryName:"\
"asc,elementCode:asc,year:desc"
```

```
    # Unzip the downloaded file
    unzip -q FAO_${1}_data.zip

    # Change original name based on last file created
    mv $(ls -rt | tail -n 1) FAO_${1}_data.csv

    # Remove the zip file (optional)
    rm FAO_${1}_data.zip

    # Prints out to the screen
    cat FAO_${1}_data.csv
fi
}
```

Problem 3

a) Download files by extension from url

```
#!/bin/sh
# Get index.html file without output
wget -q https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/

# Get the name of the .txt files and save them to a text file
egrep -oe '>[^<].*\.txt' index.html | tr -d ">" > TextList.txt

# Output current status
echo "Download starts"
# Read txt with names line by line and save it to filename var
while IFS= read -r filename
do
    # Indicates the user the file being downloaded
    echo "File ${filename} is being downloaded"

    # Download the file
    wget -q "https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/"${filename}

    # Tells the user that it has been downloaded
    echo "File ${filename} has been downloaded"

# Declare the input file for reading on while loop
done < TextList.txt

# Remove the index.html and TextList files
rm index.html TextList.txt
```