Stat 243: Advanced Shell and Pipelines

Data `AirQualityUCI.csv`

For this first part of the tutorial, please download the `AirQualityUCI.zip` file using these commands:

```
mkdir pipelines

cd pipelines

# download zip file
curl -O http://archive.ics.uci.edu/ml/machine-learning-databases/00360/AirQualityUCI.zip

# unzip
unzip AirQualityUCI.zip
```

Once unzipped, let's take a look at the first lines of the CSV file

```
head AirQualityUCI.csv
```

Extracting Columns with `cut`

To pull out vertical columns from a file you can use the `cut` command. This Unix utility operates based either on character position within the column when using the `-c` option, or on delimited fields when using the `-f` option.

Options for the `cut` command:

- `-f 1,3` Returns columns 1 and 3, delimited by tabs
- `-d ","` Use commas as the delimiters, instead of tabs; this option is used in conjuction with the `-f` option
- `-c 3-8` Returns characters 3 through 8 from the file or stream of data

With the `-c` option, numbers are given to indicate which characters to extract; with the `-f` option, the numbers indicate which columns to extract; the `-d` option indicates the type of field delimiter. By default `cut` expects tabs as the delimiter. So, for instance, to indicate a comma as a delimiter, use `-d ","`

To pull out the first column `Date` from the CSV file, you need to specify `-f 1` followed by character-delimiter flag `-c ";"`

```
# first column
cut -f 1 -d ";" AirQualityUCI.csv
```

We can just simply look at the first five lines:

```
# first lines of first column
cut -f 1 -d ";" AirQualityUCI.csv | head -n 5
```

To subset the second to fourth columns, and "save" them in a new file:

```
cut -f 2-4 -d ";" AirQualityUCI.csv > columns-2-4.csv
```

Sorting lines with `sort`

You can use the `sort` command to sort the lines of a file, or the input ppased to `sort`. By default, 'sort starts with the first character of the line and the first column of data:

```
ls | sort

cut -f 1 -d ";" AirQualityUCI.csv | sort | tail -n 5
```

Options for the `sort` command

- `-n` Sort by numeric value rather than alphabetically
- `-r` Sort in reverse order, z to a or high numbers to low numbers
- `-k 3` Sort lines based on column 3, with columns delimited by spaces or tabs
- `-t ","` Use commas for delimiters, instead of the default or tabs or white spaces
- `-u` Return only a single unique representative of repeated items

To sort based on other columns (whether separated by tabs or spaces), use the `-k` option followed by the number of the column you wish to use for sorting. Note that because values are sorted in ASCII order, blanks come alphabetically before the letter A. Another behavior is that all capital letters come before lowercase letters, so capital `Z` is alphabetically before lowercase `a`.

Sorting can proceed numerically instead of alphabetically when the `-n` option is used.

Isolating unique lines with `uniq`

Another powerful and frequently used command for extracting a subset of values from a file is `uniq`. This command removes consecutive identical lines from a file, leaving one unique representative. I order to be removes, the marching lines have to occur in immediate succession, without any intervening different lines. To get a single representative of each unique line from the entire file, in most cases you would need to first sort the lines with the `sort` command to group matching lines together.

The `uniq` command can be used with the `-c` option to count the number of occurrences of a line or value.

Options for the `uniq` command:

- `-c` Counts the number of occurrences of each unique line
- `-f 4` Ignore the first 4 fields (columns delimited by any number of spaces) in determining uniqueness
- `-i` Ignores cases when determining uniqueness

Extracting particular rows from a file

What if you want to extract those line in `AirQualityUCI.csv` starting with the value `10/03/2004`? The command `grep` is a tool that quickly extracts only those lines of a file that match a particular regular expression.

```
grep "10/03/2004" AirQualityUCI.csv
```

The first argument, `"10/03/2004"`, is the regular expression, and the second argument specifies the source file you want it to examine. The `grep` program scans the file and displays only those lines that contain the search phrase. We need to use quotes around the regular expression as a good practice.

In the previous example, the results were simply sent to the screen. But you can redirect then to a file `air-quality-10-03-2004.csv`

```
grep "10/03/2004" AirQualityUCI.csv > air-quality-10-03-2004.csv
```

Now you have a file that is a subset of the original, containing only those lines with date `"10/03/2004"`. The only issue now is that the new file doesn't have a header. To solve this, you can do something like this:

```
# redirect the header
head -n 1 AirQualityUCI.csv > air-quality-10-03-2004.csv

# append the lines with "10/03/2004"
grep "10/03/2004" AirQualityUCI.csv >> air-quality-10-03-2004.csv
```

Options that modify the behavior of `grep`

- `-c` Show only a count of the results in the file
- `-v` Invert the search and show only lines that do NOT match
- `-i` Match without regard to case
- `-E` Use regular expression syntax "Extended" regex
- `-l` List only the file names containing matches
- `-n` Show the line numbers of the match
- `-h` Hide the filenames in the output

Bash Shell Examples

Here we'll work through a few examples to start give you a feel for using the bash shell to manage your workflows and process data.

First, let's download the data file `cpds.csv` (Comparative Political Data Set) from the `data/` folder in the github repo:

```
curl -O https://raw.githubusercontent.com/berkeley-stat243/stat243-fall-2014/master/data/cpds.c
```

Our first mission is some basic manipulation of a data file. Suppose we want to get a sense for the number of countries in the file `cpds.csv`

```
# looking at the second column 'country'
cut -f 2 -d "," cpds.csv | less

# first few rows (without header)
cut -f 2 -d "," cpds.csv | tail -n +2 | head

# unique countries
cut -f 2 -d "," cpds.csv | tail -n +2 | uniq

# number of unique countries
cut -f 2 -d "," cpds.csv | tail -n +2 | uniq | wc -l
```

Our second mission: How can we count the number of fields (i,e, "columns") in a file?

```
# header
head -n 1 cpds.csv

# detecting commas
head -n 1 cpds.csv | grep ","

# printing only matching commas
head -n 1 cpds.csv | grep -o ","

# counting number of commas
head -n 1 cpds.csv | grep -o "," | wc -l
```

Our third mission: How can we subset the lines corresponding to 1960 in a new file?

```
grep 1960 cpds.csv | less

grep 1960 cpds.csv > cpds-1960.csv

# including header
head -n 1 cpds.csv > cpds-1960.csv
grep 1960 cpds.csv >> cpds-1960.csv
```

Our fourth mission: How can we subset the lines for the years 1960, 1970, 1980, 1990, 2000, and 2010, programatically? This where a for-loop comes very handy.

```
for year in 1960 1970 1980 1990 2000 2010
do
  echo subsetting file for year $year
  head -n 1 cpds.csv > cpds-$year.csv
  grep $year cpds.csv >> cpds-$year.csv
done

ls
```

Alternatively, we can to something like this:

```
for (( year=1960; year<=2010; year+=10 ))
do
  echo subsetting file for year $year
  head -n 1 cpds.csv > cpds-$year.csv
  grep $year cpds.csv >> cpds-$year.csv
done
```

And if you want to subset for all the available years:

```
for (( year=1960; year<=2010; year+=1 ))
do
  echo subsetting file for year $year
  head -n 1 cpds.csv > cpds-$year.csv
  grep $year cpds.csv >> cpds-$year.csv
done
```

Our fifth mission: How can we subset the lines for each country in a separate file programatically?

```
# again, here are the names of the countries
cut -f 2 -d "," cpds.csv | tail -n +2 | uniq

# let's create a variable with those names
countries=$(cut -f 2 -d "," cpds.csv | tail -n +2 | uniq)
echo $countries
```

The issue here are the quotes around the names. Here's one option to get rid of those quotes. First, let's save the names in a text file `countries.txt`:

```
cut -f 2 -d "," cpds.csv | tail -n +2 | uniq > countries.txt
```

We haven't talked about the command `sed` but here's how to use it to substitute all quotes by empty characters:

```
sed 's/"//g' countries.txt
```

Now we need to save those names in our variables `countries`

```
countries=$(sed 's/"//g' countries.txt)

echo $countries
```

And finally we can use a for-loop:

```
for country in $countries
do
  echo subsetting for country $country
  head -n 1 cpds.csv > $country.csv
  grep $country cpds.csv >> $country.csv
done
```

There is one more issue in this case though: where's the file `New Zealand.csv`? What happened in this case?

Challenges

1. How can we fix the issue with New Zealand's file? Here's one suggestions: use `sed` to strip out the quotation marks and spaces in `cpds.csv`. Then re-do the loop.
2. Determine the minimum unemployment value (field #6) in `cpds.csv` for *Belgium*.
3. Figure out how to manipulate challenge 2 to do the calculation for all the countries and print the results to the screen?
4. How would you instead store the results in a file?

Solution challenge 1:

```
# strip out quotes with redirection
sed 's/"//g' cpds.csv > cpds-copy.csv

# in-place substitution of blank space
sed -i '' 's/ //g' cpds-copy.csv

# examine countries
cut -f 2 -d "," cpds-copy.csv | tail -n +2 | uniq
```

Solution challenge 2:

```
grep Belgium cpds.csv | cut -f 6 -d "," | sort | head -n 1
```

Solution challenge 3:

```
countries=$(cut -f 2 -d "," cpds-copy.csv | tail -n +2 | uniq)

for country in $countries
do
  echo minimum unemployment value for $country
  grep $country cpds-copy.csv | cut -f 6 -d "," | sort | head -n 1
  echo
done
```

Solution challenge 4:

```
touch unemployment.txt

countries=$(cut -f 2 -d "," cpds-copy.csv | tail -n +2 | uniq)

for country in $countries
do
  echo minimum unemployment value for $country >> unemployment.txt
  grep $country cpds-copy.csv | cut -f 6 -d "," | sort | head -n 1 >> unemployment.txt
  echo
done
```