

# **Large Datasets in Shell Scripting**

Hannah Castelvechi

Pawprint: hgc8zc

IT/CS UNIX Operating Systems

## TOPIC

My chosen topic is “Using shell scripting along with an original dataset to obtain some insight into a large volume of data”. As a computer science major, many of my courses have dealt with extensive datasets and file manipulation within the C programming language. I chose this topic because I am interested in learning the benefits of using shell scripting to obtain insight into large quantities of data. Understanding the effectiveness of using shell scripting will be beneficial in my future career as most of my work will involve managing data. Learning how to consider statistics or trends within a file using shell scripting as opposed to languages like C/C++ may help with efficiency in ways that are to be discussed further in the upcoming sections.

## RESEARCH

The scope of this course includes techniques on file manipulation such as sorting data based on numerical fields, locating repeating lines, and replacing strings or characters. The capabilities of shell scripting in regards to large datasets expands to complex analysis of patterns and statistics on files and even outside sources like websites or articles. For example, if a Twitter user wanted to predict whether a direct message was sent from a spam account, the user could download their messages using The Twitter API project and build a shell script to analyze the data ([Using Bash for Data Pipelines. Using bash scripts to create data... | by Elliott Saslow](#)). The user could analyze the data from the direct messages by locating repeating words or phrases in order to locate patterns that indicate messages from spam users. Some example phrases are “free”, “guarantee”, “million”, “call now”, and various others ([Common Things That Trigger Spam Filters](#)). Within the shell script, the user could use the “grep” command to locate these words and phrases within their messages. Furthermore, this type of analysis can be applied to sources like CSV files that contain fields of data as opposed to blocks of text. Lines in a file may

be organized by a specific field's alphabetical position, numerical value, or a multitude of other distinctions. Furthermore, a file may be read by a shell script in a way in which more intuitive data is outputted. For example, a file containing rows and columns of data may be reworked so that only the first column is displayed using the “cut” command. Similarly, values within specific fields may be stored within variables so that a script may use a loop to print out desired pieces of data ([Shell - Read a text or CSV file and extract data](#)). This example will be further demonstrated in the Application section as applied to an original file of data that I will analyze using these shell scripting techniques.

Applying these methods of analysis using shell scripting is effective in that, by comparison to languages such as C or C++, they are highly portable and readable. Whereas a shell script may be transferred between different unix operating systems, a C/C++ program must be rebuilt and “it may not work as expected if it uses architecture-specific code” ([What Is Shell Scripting and Why You Should Use It](#)). Additionally, shell scripting is easier to use when administrative tasks like redirecting output, and the scripts can be more easily debugged since the user has access to the source code. Overall, shell scripting is more transparent and therefore efficient when it comes to accessing large datasets and has a multitude of capabilities to analyze them.

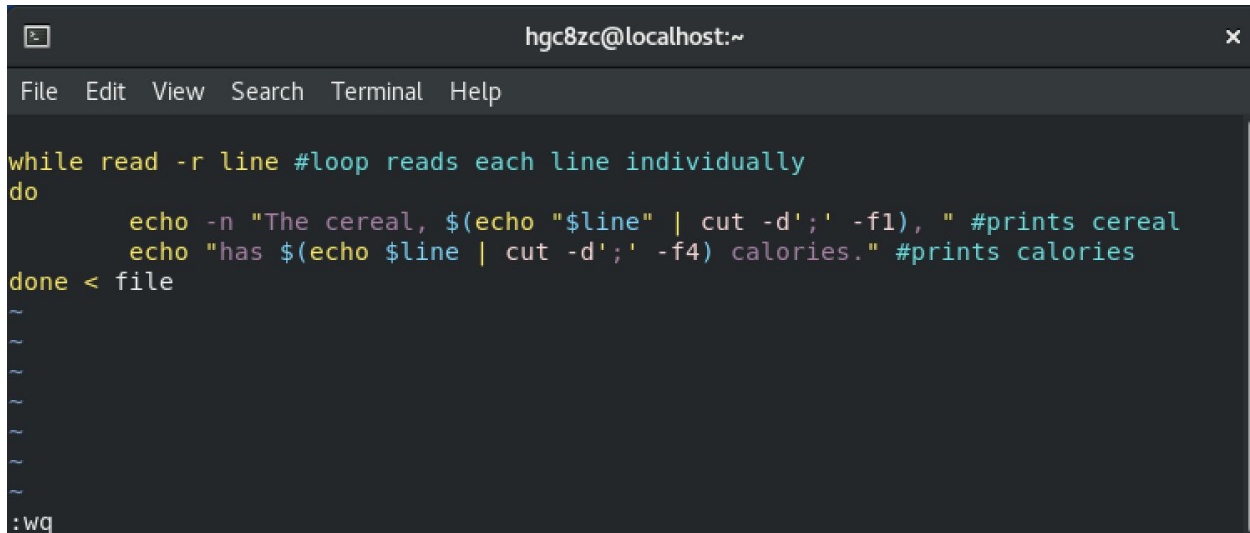
## APPLICATION

To demonstrate the previously described methods of data analysis in shell scripting, an example CSV file, cereal.csv, will be used for testing. The file displays nutritional data for 77 popular cereals beginning with a two-line heading. A sample output of the file is as follows:

name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
String	Categorical	Categorical	Int	Int	Int	Int	Float	Float	Int	Int	Int	Int	Float	Float	Float
100% Bran	N	C	70	4	1	130	10	5	6	280	25	3	1	0.33	68.402973
100% Natural Bran	Q	C	120	3	5	15	2	8	8	135	0	3	1	1	33.983679
All-Bran	K	C	70	4	1	260	9	7	5	320	25	3	1	0.33	59.425505
All-Bran with Extra Fiber	K	C	50	4	0	140	14	8	0	330	25	3	1	0.5	93.704912
Almond Delight	R	C	110	2	2	200	1	14	8	-1	25	3	1	0.75	34.384843

\*The entirety of the file is located within the page titled [Project datasets](#) in Sources

If the user was interested in determining which cereal has the most nutritional value, there are a multitude of ways in which shell scripting can be utilized. The first example, in which the user wishes to display each Cereal and their corresponding calorie amounts can be done by the following shell script followed by its output:



```

hgc8zc@localhost:~
File Edit View Search Terminal Help

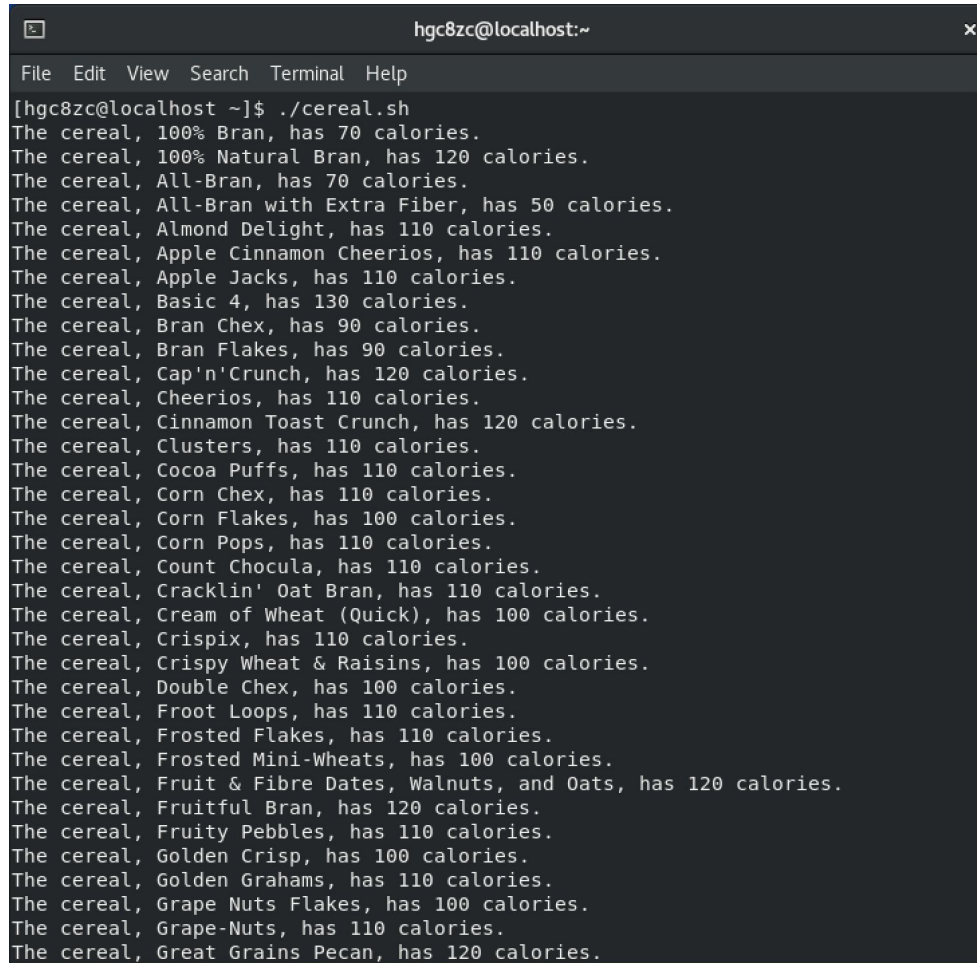
while read -r line #loop reads each line individually
do
    echo -n "The cereal, $(echo "$line" | cut -d';' -f1), " #prints cereal
    echo "has $(echo $line | cut -d';' -f4) calories." #prints calories
done < file
~
~
~
~
~
~
~
:wq

```

#### \*Example 1 Shell Script

The script above uses a “while-loop” to traverse through the lines of the file, disregarding the two header lines, and singles out the fields corresponding to the cereal’s name and its calories. In this case, these are the first and fourth fields which are distinguished by the semicolon delimiters. This is done using the “cut” command twice for each line of the file.

The output, appearing on the following page, displays the data for each line in the format “The cereal, \_\_\_\_\_, has \_\_\_\_\_ calories” as done by the “echo” commands along with “cut” to print out the desired data from each line.



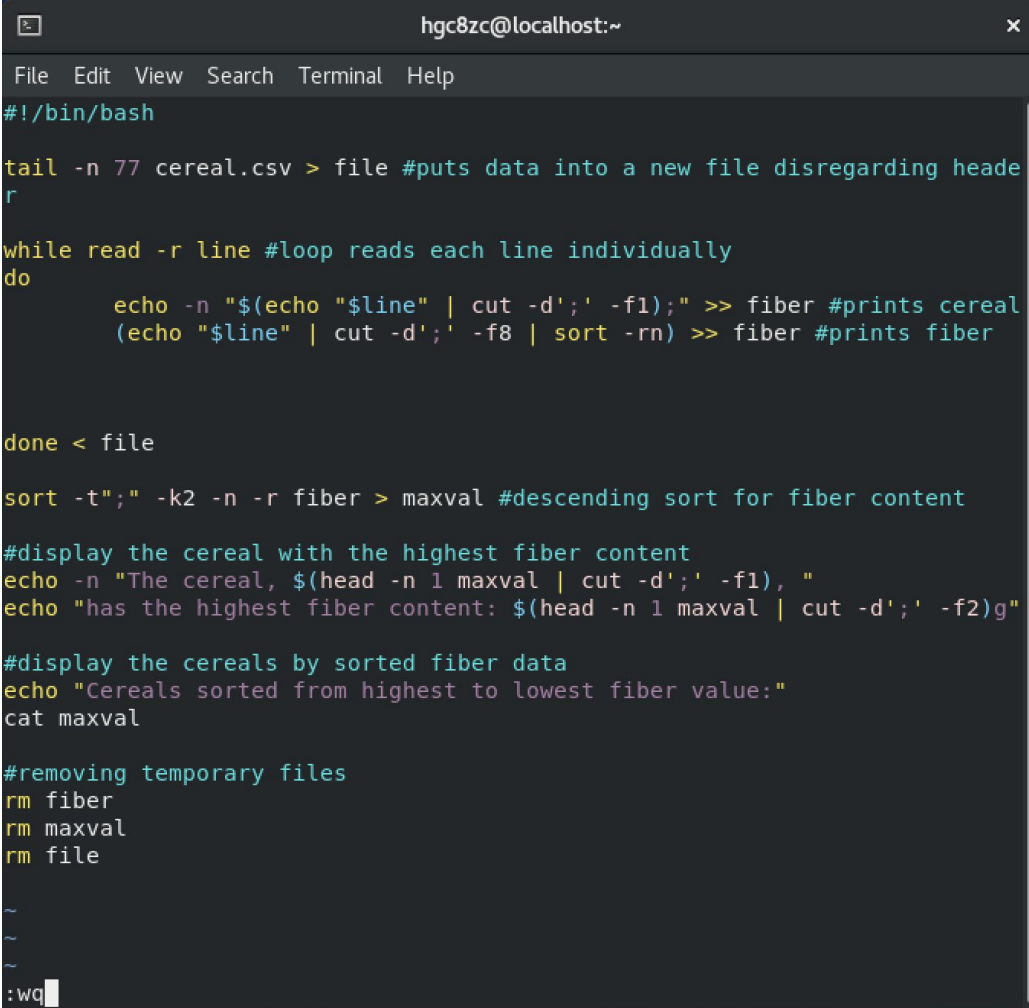
```

hgc8zc@localhost:~
File Edit View Search Terminal Help
[hgc8zc@localhost ~]$ ./cereal.sh
The cereal, 100% Bran, has 70 calories.
The cereal, 100% Natural Bran, has 120 calories.
The cereal, All-Bran, has 70 calories.
The cereal, All-Bran with Extra Fiber, has 50 calories.
The cereal, Almond Delight, has 110 calories.
The cereal, Apple Cinnamon Cheerios, has 110 calories.
The cereal, Apple Jacks, has 110 calories.
The cereal, Basic 4, has 130 calories.
The cereal, Bran Chex, has 90 calories.
The cereal, Bran Flakes, has 90 calories.
The cereal, Cap'n'Crunch, has 120 calories.
The cereal, Cheerios, has 110 calories.
The cereal, Cinnamon Toast Crunch, has 120 calories.
The cereal, Clusters, has 110 calories.
The cereal, Cocoa Puffs, has 110 calories.
The cereal, Corn Chex, has 110 calories.
The cereal, Corn Flakes, has 100 calories.
The cereal, Corn Pops, has 110 calories.
The cereal, Count Chocula, has 110 calories.
The cereal, Cracklin' Oat Bran, has 110 calories.
The cereal, Cream of Wheat (Quick), has 100 calories.
The cereal, Crispix, has 110 calories.
The cereal, Crispy Wheat & Raisins, has 100 calories.
The cereal, Double Chex, has 100 calories.
The cereal, Froot Loops, has 110 calories.
The cereal, Frosted Flakes, has 110 calories.
The cereal, Frosted Mini-Wheats, has 100 calories.
The cereal, Fruit & Fibre Dates, Walnuts, and Oats, has 120 calories.
The cereal, Fruitful Bran, has 120 calories.
The cereal, Fruity Pebbles, has 110 calories.
The cereal, Golden Crisp, has 100 calories.
The cereal, Golden Grahams, has 110 calories.
The cereal, Grape Nuts Flakes, has 100 calories.
The cereal, Grape-Nuts, has 110 calories.
The cereal, Great Grains Pecan, has 120 calories.

```

Displaying the data this way would allow for the user to read through the calorie values of each cereal in a way that is easier to conceptualize than the original CSV file. However, the user could take this a step further by singling out a cereal with the most prominent data in a specific category. For example, the cereal with the highest fiber content.

The following shell script takes the data from the CSV file and inputs only the field containing cereal names and their corresponding fiber values. The file, only containing two fields, is sorted so that the cereals are listed from highest to lowest fiber content. This method can be used for any of the numerical categories of this file in order to obtain their extreme values as well as the sorted values in between. The shell script and its output are as follows:



```

hgc8zc@localhost:~
File Edit View Search Terminal Help
#!/bin/bash

tail -n 77 cereal.csv > file #puts data into a new file disregarding header

while read -r line #loop reads each line individually
do
    echo -n "$(echo "$line" | cut -d';' -f1);" >> fiber #prints cereal
    (echo "$line" | cut -d';' -f8 | sort -rn) >> fiber #prints fiber

done < file

sort -t";" -k2 -n -r fiber > maxval #descending sort for fiber content

#display the cereal with the highest fiber content
echo -n "The cereal, $(head -n 1 maxval | cut -d';' -f1), "
echo "has the highest fiber content: $(head -n 1 maxval | cut -d';' -f2)g"

#display the cereals by sorted fiber data
echo "Cereals sorted from highest to lowest fiber value:"
cat maxval

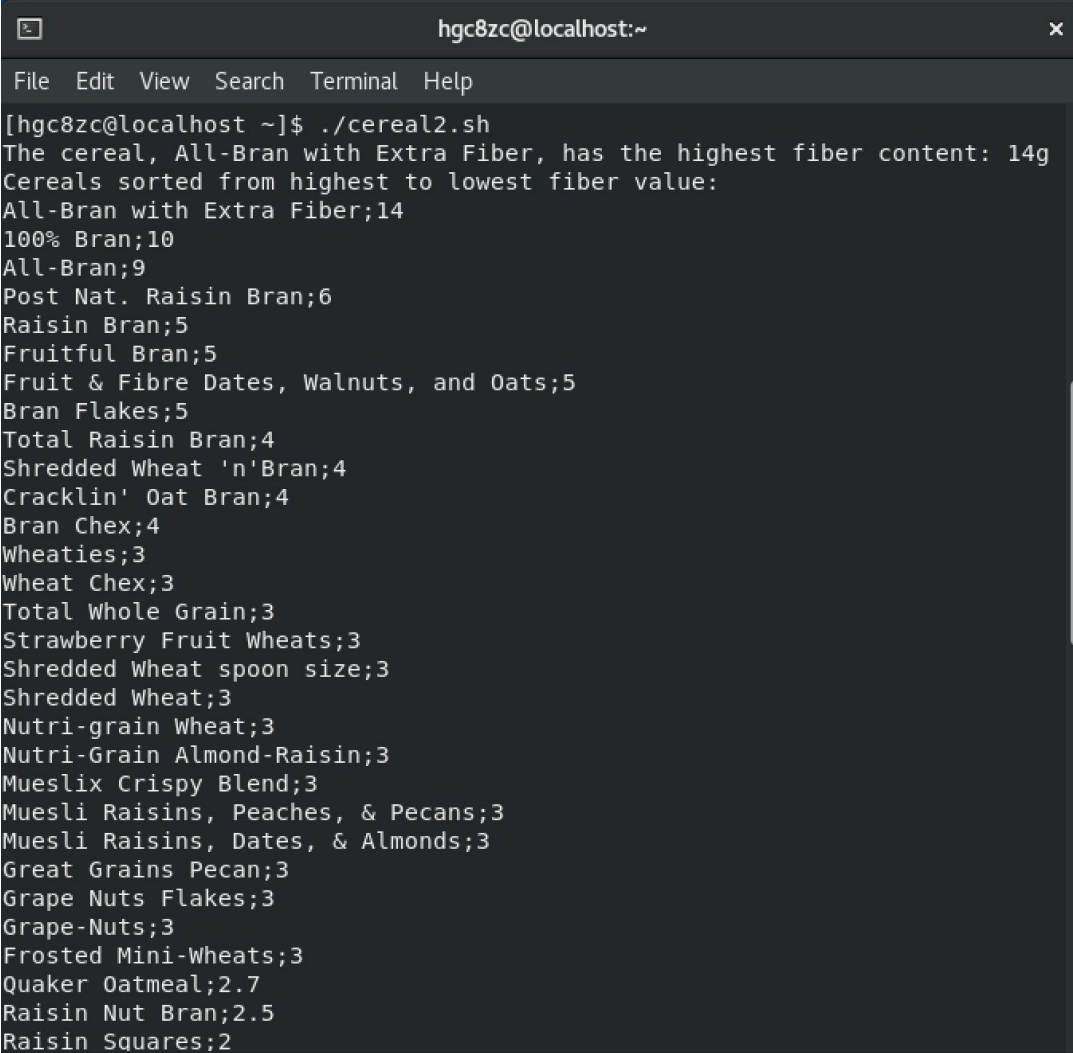
#removing temporary files
rm fiber
rm maxval
rm file

~
~
~
:wq

```

### \*Example 2 Shell Script

The above script takes elements from Example 1, specifically in the first for-loop that isolates the desired fields of data. It goes beyond simply printing them by sorting the cereals by a specified numerical field using the “sort” command. Furthermore, if the user wanted to simply display the maximum or minimum value along with its cereal, the user could input the sorted data into a file and then use either the “head” or “tail” command. I wanted to display the cereal with the highest fiber content, and since my file is sorted in descending order, I used the “head” command that printed, to no surprise, “All-Bran with Extra Fiber”. The sample output appears on the following page.



```

hgc8zc@localhost:~
File Edit View Search Terminal Help
[hgc8zc@localhost ~]$ ./cereal2.sh
The cereal, All-Bran with Extra Fiber, has the highest fiber content: 14g
Cereals sorted from highest to lowest fiber value:
All-Bran with Extra Fiber;14
100% Bran;10
All-Bran;9
Post Nat. Raisin Bran;6
Raisin Bran;5
Fruitful Bran;5
Fruit & Fibre Dates, Walnuts, and Oats;5
Bran Flakes;5
Total Raisin Bran;4
Shredded Wheat 'n'Bran;4
Cracklin' Oat Bran;4
Bran Chex;4
Wheaties;3
Wheat Chex;3
Total Whole Grain;3
Strawberry Fruit Wheats;3
Shredded Wheat spoon size;3
Shredded Wheat;3
Nutri-grain Wheat;3
Nutri-Grain Almond-Raisin;3
Mueslix Crispy Blend;3
Muesli Raisins, Peaches, & Pecans;3
Muesli Raisins, Dates, & Almonds;3
Great Grains Pecan;3
Grape Nuts Flakes;3
Grape-Nuts;3
Frosted Mini-Wheats;3
Quaker Oatmeal;2.7
Raisin Nut Bran;2.5
Raisin Squares;2

```

This script first displays the cereal with the highest fiber content as well as the sorted data that lists the name of the cereal followed by its content of fiber in grams.

Though statistics regarding cereal may be trivial in comparison to real world issues, using shell scripting is extremely valuable in obtaining insight into any and all large datasets. Using shell commands along with a unix/linux environment is fairly transparent compared to using complex programming languages, and there is no shortage of capabilities.

## SOURCES

In order of use in report

- <https://towardsdatascience.com/using-bash-for-data-pipelines-cf05af6ded6f>
- [https://knowledgebase.constantcontact.com/articles/KnowledgeBase/5649-common-phrases-that-trigger-spam-filters?+target=&lang=en\\_US](https://knowledgebase.constantcontact.com/articles/KnowledgeBase/5649-common-phrases-that-trigger-spam-filters?+target=&lang=en_US)
- <https://www.theunixschool.com/2012/05/shell-read-text-or-csv-file-and-extract.html>
- <https://www.makeuseof.com/tag/what-is-shell-scripting/>
- <https://perso.telecom-paristech.fr/eagan/class/igr204/datasets> (for CSV file)