Adventures in Data Science with Bash

By *Robert Aboukhalil*

Introduction

Following the 2008 US elections, all eyes turned to Nate Silver. Using polls and clever number-crunching, the former baseball analyst wowed the public by correctly predicting the election outcomes in 49 out of 50 states. In Indiana—the only state he miscalled—the race was very tight, with only a 1% margin of victory separating the two candidates.

Unlike most political pundits, data scientists such as Nate Silver, the Princeton Election Consortium (49/50 correct) and Electoral-Vote.com (48/50 correct) use data and a good dose of statistical modeling to forecast elections with great accuracy. In doing so, they have put data science on the map for the general public.

Why this book?

As it turns out, you don't have to be a statistician to perform such analyses. Increasingly, data science is occupying a greater part of our lives and our work. Whether you are a biologist, journalist, lawyer, or financial analyst, the ability to analyze data to quickly answer a question is a powerful skill to have, and it is this skill that this book aims to develop.

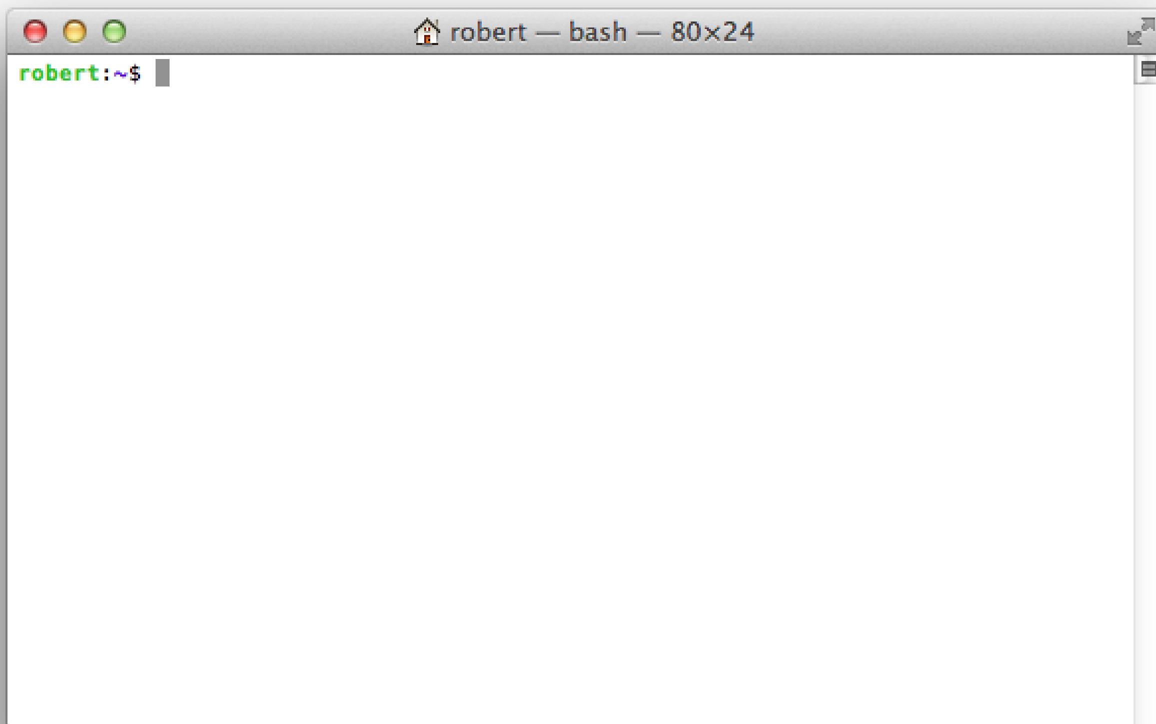
In Chapter 0, we introduce the basics of the command line that are necessary to get started. In each subsequent chapter lies an adventure that starts with questions that we would like answered. Step by step, we will guide you through answering these questions using only command-line tools. For example, we'll investigate the eating habits of Chipotle customers, calculate the average tip of a NYC cab driver, determine how often flights arrive on time, and more.

Shall we get started?

Adventure 0: Baby steps

What is the command line?

The adventures in this book revolve around using the *command line*, which is simply a window where you type commands that your computer performs:



Much like a graphical calculator, the command line allows you to input commands, see the result, and see a history of previous commands entered. In the command line (also known as a *terminal*), you will find very little clicking, dragging, menus or any other meaningful user interface; it's all about typing commands.

These commands range from basic file operations *(create a folder, delete a file)*, to more interesting operations for our data science adventures *(sort a file by a given column, extract the lines of a file that match a certain pattern)*.

Using scripting languages such as *Bash*, the command line allows you to perform actions that would otherwise be very tedious to perform. Before we start with our adventures, let's study a sample *Bash* script that solves a real-life problem.

Say you returned from a trip to Peru, where you took hundreds of pictures with your phone. In fact, you took 716 pictures. Unfortunately, when these pictures are downloaded to your computer, they are given cryptic file names, such as `DSC_3018.JPEG`. To rename the files so that they follow the pattern `PERU_3018.JPEG`, there are two approaches: **(1)** renaming 716 files by hand (please don't try this at home); or **(2)** running the following *Bash* command, which would take a fraction of a second to complete:

```
for file in *.JPEG; do
    mv ${file} ${file/DSC/PERU}
done
```

Don't worry if you don't understand what any of this means, you're not supposed to yet! But to give you an idea for what this script does, here's how to interpret the code line by line:

- **Line 1**: For every .JPEG file in my folder, do the following:
- **Line 2**: Rename that file by replacing `DSC` with `PERU` in the filename.
- **Line 3**: Once all files are renamed, we're all done

Even if this didn't entirely make sense, just know that the command line offers a much quicker way to perform such operations that would have taken a very long time to do manually.

Opening the command line

Now that we know what the command line is and have seen an example of what it can do, let's open the command line on your computer. Depending on your operating system, your command line will be found in different places:

If you have a Mac OS machine

Since Mac OS is UNIX-based, it is relatively easy to launch the command line. You have two options:

- Open *Finder*, navigate to the `Applications` folder, open the `Utilities` folder, and launch the program called `Terminal`
- A quicker way is to perform a Spotlight search for "Terminal" (usually, the keyboard shortcut for opening Spotlight is `⌘ Command` + `Space`)

If you have a Linux machine

On a Linux machine such as Ubuntu, you have two options:

- Click on `Applications`, point to `Accessories`, and launch the `Terminal` program
- Press `Ctrl + Alt + T`

If you have a Windows machine

It is slightly more difficult to get Bash running on Windows, but let me guide you through each step. You'll have to

install a program called *Cygwin*, which will essentially emulate the functionality of the command line:

1. To find out if you have a 32-bit or 64-bit version of Windows, go to this link, and scroll down to *Automatic version detection results*. Look for whether it says *32-bit* or *64-bit* and keep that in mind.
2. Next, go to the Cygwin installation page. If you have a 32-bit Windows, click on *Run setup-x86.exe*. If you have a 64-bit Windows, click on *Run setup-x86_64.exe*.
3. Once downloaded, follow all the steps to install *Cygwin* (say *Yes* to having a *Cygwin* icon on your Desktop).
4. Go to your desktop and double-click on *Cygwin Terminal*.
5. You're all set!

Getting to know the command line

Listing the files on your Desktop

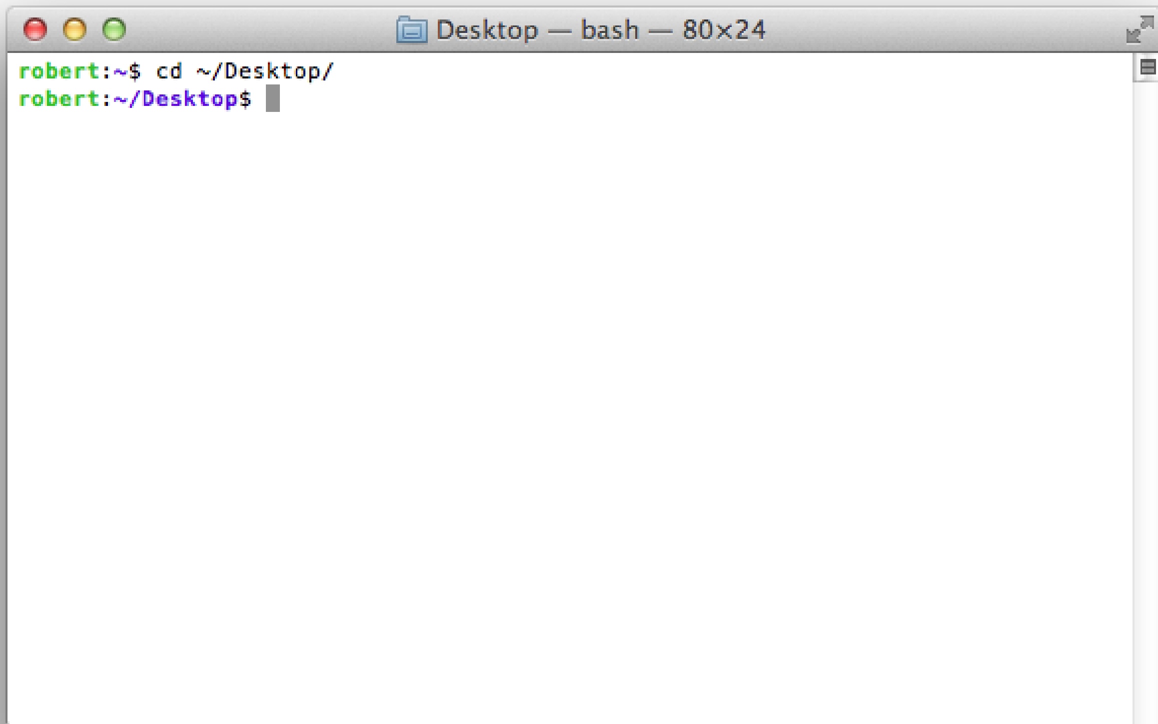
When you work inside *Finder* or *Windows Explorer*, you're working within one folder at a time. The same is true of the command line: all commands you execute are executed relative to your current folder. For example, if you ask the command line to delete the file `test.txt`, it will delete the file `test.txt` within the current directory. If you ask the command line to create a file, it will be created in the current directory.

The first thing we will do on the command line is to navigate to our Desktop folder and list the files, just like we would in *Finder* or *Windows Explorer*. To navigate to your Desktop, use the `cd` command (change directory) by typing the following and pressing Enter:

```
cd ~/Desktop/
```

The `~` is simply a shortcut that the command line understands as meaning your home folder. For MacOS, that means `/Users/robert`, and for Linux/Cygwin, it usually means `/home/robert`.

If you typed the command above correctly, you should find yourself in your Desktop folder:



Note that Bash did not output anything on the screen because it did not have to, which brings us to an important principle of command line scripting:

> **PRINCIPLE**: In the command line world, no news is generally good news, unless you ask for information or if an error occurs.

Now that we're in our Desktop folder, we can list all the files that are present there using the `ls` command (list):

```
ls
```

Creating a folder for future analyses

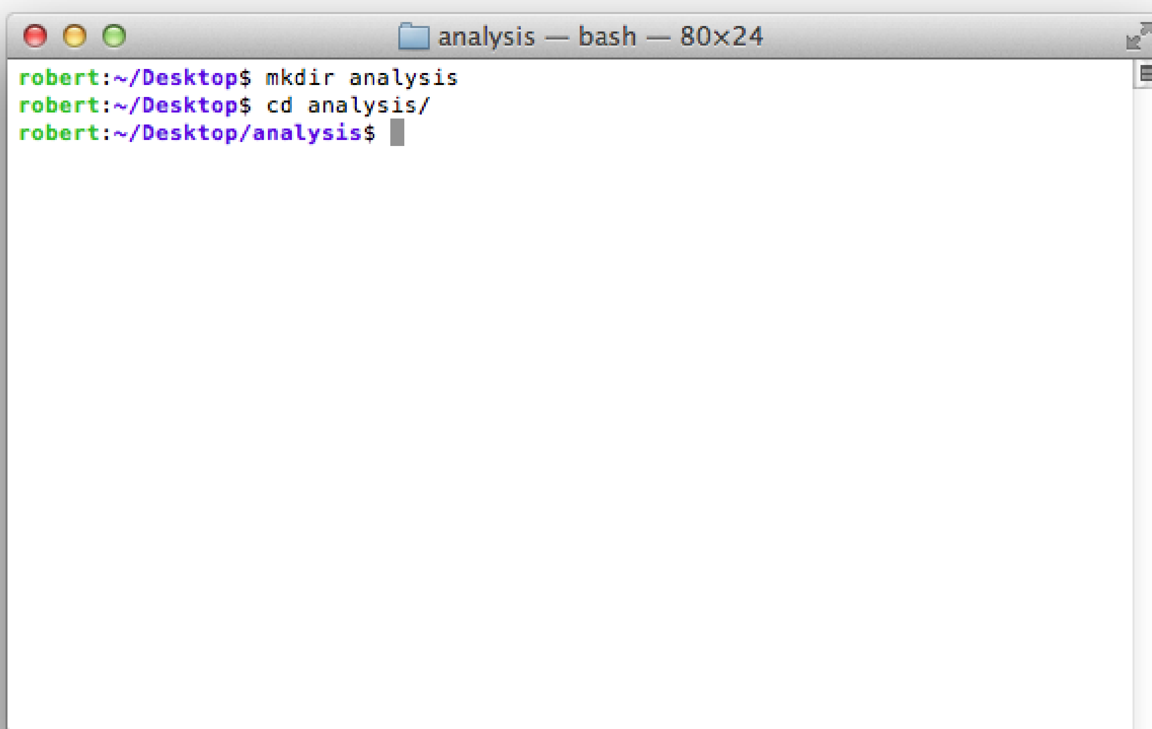
Next, we'll create a folder on your Desktop where we will keep all the analysis we do from each adventure in Chapter 1 onward. To do so, use the `mkdir` (make directory) command as follows:

```
mkdir analysis
```

To navigate to your new folder, use `cd` (change directory) once more, but this time with a small twist. Instead of typing `cd analysis`, only type `cd analy` and press the `Tab` key on your keyboard. What should happen is that Bash will autocomplete to `cd analysis/`. This is a very powerful feature that will save you a lot of time in the future.

> **TIP**: To autocomplete on the command line, use the `Tab` key on your keyboard. If there are more than 1 result, pressing `Tab` will instead output all the matches.

At this point, here's what you should see:



Before we end, one last protip:

> **TIP:** To see and re-run previously-entered commands, use the `Up` and `Down` keys on your keyboard.

Summary

Congratulations, you made it through Chapter 0! Despite the unfortunate numbering of this chapter, you have reached an important milestone in learning Bash scripting, and are now ready to face the adventures that follow.

Review

- `ls` : list files and folders in the current directory
- `cd folder` : change current directory to the folder called `folder`
- `mkdir newfolder` : create a subfolder called `newfolder` under the current folder
- `~` : shorcut notation that denotes your home directory

Protips

- No news is good news on the command line, except when asking for output (as we did with `ls` and `pwd`), or in case of error
- To autocomplete on the command line, use the `Tab` key on your keyboard. If there is one match, Bash will autocomplete to that match; if there are more matches, they will all be listed.
- To see, edit and re-run previously-entered commands, use the `Up` and `Down` keys on your keyboard.

Adventure 1 – Eating habits at Chipotle

It's a dreary Monday evening and you are driving home, when suddenly hunger sets in. You notice a Chipotle up ahead so you take the next right. Inside, you order a steak burrito and sit to have a bite. While munching on your burrito, you cannot help but wonder: How many other fellow customers prefer the steak burrito over the chicken burrito?

That is our first adventure!

Before we go any further, let's setup our working environment by creating a subfolder within the `analysis` folder on our Desktop, where we will store our analysis files. To do so, let's first fire up a command line and navigate to our `analysis` folder:

```
cd ~/Desktop/analysis/
```

Once there, create a new sub-folder called `chapter1`:

```
mkdir chapter1
```

Now that we have our sub-folder, let's move into it:

```
cd chapter1/
```

Next, we need the data. Luckily, in February of 2015, the New York Times' *TheUpshot* column published an article that calculate the typical number of calories in a Chipotle order. To do so, they obtained data from 1,800 Grubhub orders from July to December 2012, and that data was made publicly available. Right-click that link, and choose *Save Link As*, and save it as `orders.tsv` in the folder on your Desktop `analysis/chapter1`.

Now that we're ready, lettuce get started!

Sneak peak

This dataset is very small and we could in principle open it in a text editor or in Excel. However, datasets are often larger and cumbersome to open in their entirety. Instead, let's get a sneak peak of the data. This is often the first thing to do when you get your hands on new data; previewing it is important to get a sense for what it contains, how it is organized, and whether the data makes sense in the first place.

To help us get a preview of the data, we can use the command `head`, which as the name suggests, shows the first few lines of a file:

```
head "orders.tsv"
```

You should see the first 10 lines of the file output onto the screen:

chapter1 — bash — 80×24

```
robert:~/Desktop/analysis/chapter1$ head "orders.tsv"
order_id	quantity	item_name	choice_description	item_pri
ce
1	1	Chips and Fresh Tomato Salsa	NULL	$2.39
1	1	Izze	[Clementine]	$3.39
1	1	Nantucket Nectar	[Apple]	$3.39
1	1	Chips and Tomatillo-Green Chili Salsa	NULL	$2.39
2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans,
Rice, Cheese, Sour Cream]]	$16.98
3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sour
Cream, Guacamole, Lettuce]]	$10.98
3	1	Side of Chips	NULL	$1.69
4	1	Steak Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables,
Black Beans, Pinto Beans, Cheese, Sour Cream, Guacamole, Lettuce]]	$11.75
4	1	Steak Soft Tacos	[Tomatillo Green Chili Salsa, [Pinto Bea
ns, Cheese, Sour Cream, Lettuce]]	$9.25
robert:~/Desktop/analysis/chapter1$
```

To see more than the first 10 lines, e.g. the first 25, use the `-n` option:

```
head -n 25 "orders.tsv"
```

Here, `"orders.tsv"` is a **command-line argument** that is given to `head` and the `-n` is an option of the `head` command, which allows us to overwrite the 10-line default. Such **command-line options** are typically written as a dash followed by a string, a space, and the value of the option (e.g. `-n 25`). Generally, they are placed the closest to the command name. In some cases, when the options don't require a value but instead are made for toggling a feature on or off, simply write the option on it's own (e.g. `top -h` shows the help page for the command `top`).

> **EXTRA CREDIT**: For some commands, the options are written using two dashes and words (`e.g. bash --version`), and some commands will accept both.

From the first 25 lines of the file, we can infer that the data is formatted as a file with Tab-Separated Values (which was hinted at by the *TSV* file extension). From the first line (often called a **header line**) and the first few lines of data, we can infer the column names:

1. ID of an order
2. Quantity of item bought
3. Item name
4. Description of the condiments (e.g. black beans, rice, etc.)
5. The price of the item

How many Steak Burritos were ordered?

Step 1: Find the Steak Burritos

To list all the lines in the data file that contain the phrase "Steak Burrito", we need to introduce the command `grep` (*global regular expression print*). In a nutshell, `grep` allows you to look through all lines in a file and only output those that correspond to a pattern. In our case, we want to find all the lines in `orders.tsv` that contain "Steak Burrito". Here's how we do it:

```
grep "Steak Burrito" "orders.tsv"
```

Here, the `grep` command takes two **command-line arguments**: the first is the pattern, and the second is the file in which we want to search for this pattern. If you run this command you should see hundreds of lines scroll down your screen, corresponding to only the lines in `orders.tsv` that contain "Steak Burrito".

Now that we have all the lines in the file that contain "Steak Burrito", all we need to do is count how many such lines there are.

Step 2: Counting lines

To do this step, we need to introduce the concept of **pipes**, which are one of the most powerful features of command-line scripting.

Pipes are represented by the vertical bar symbol `|`. Essentially, a pipe allows you to give the output of one command to the input of another command, without having to save the intermediate output to a file.

For example, say you were in your Peru folder with 716 pictures. Typing `ls` in that folder would be inconvenient because there are many files to list. To instead output the first 10 files in that folder, we can use `ls | head`. Here, the output of `ls` is not shown on screen, but is given to the `head` command, which will only output the first 10 lines of the input.

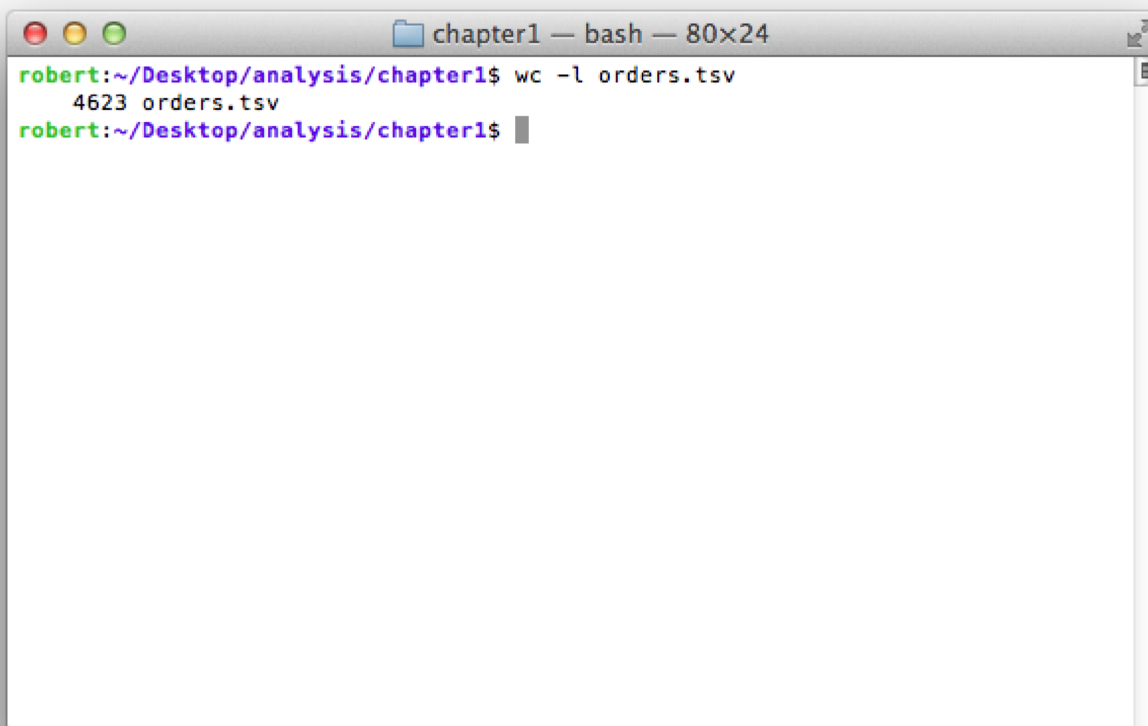
For this adventure, we would like to give the output of `grep` to another command which can tell us how many lines that output has.

As it turns out, there is a command that will let us do just that: `wc`. `wc` stands for *word count* and is used to count how many words are in a given input, but is most often used with the `-l` option that will instead count how many lines are in an input.

So let's start by counting how many lines total there are in the `orders.tsv` file:

```
wc -l "orders.tsv"
```

You should see this output:



Great, now let's use pipes and bring everything we learned together:

```
grep "Steak Burrito" "orders.tsv" | wc -l
```

Again, the output of grep–the lines in the file that contain "Steak Burrito"–is passed on to `wc -l`, which will count the number of lines in the file. Notice that here we didn't specify a filename in the `wc -l` command; we didn't have to because we piped our data into `wc -l`, so it didn't need to read the data from a file.

Running this command should give you the output of `368`. So for this dataset, 368 steak burritos were ordered. What about chicken burritos?

All we need to do now is to run the same command but replace "Steak" with "Chicken":

```
grep "Chicken Burrito" "orders.tsv" | wc -l
```

You should see 553. We conclude that for this particular dataset, customers ordered 1.5 times more chicken burrito than steak burrito (of course we have to be careful about generalizing: our result might only be valid for this dataset, time, day, location, etc.)

> **EXTRA CREDIT**: Find out how often customers ordered a "Veggie Burrito" (95), "Barbacoa Burrito" (91 orders), and "Carnitas Burrito" (59).

Guacamole with your chips?

As you finish eating your burrito and start dipping your tortilla chips in Guacamole, another question comes to mind: What kind of salsa do most customers get with their chips?

Of course, we could use the same command as shown above and count how many of each type of salsa we find. However, what if you did not know ahead of time about the various kinds of salsas that are available at Chipotle? In this section, we will see how to infer this from the data.

Extract the lines with Chips and Salsa

The first step, as above, is to find the lines in `orders.tsv` that contain orders of chips and salsa. To find those, we can use `grep` as follows:

```
grep "Chips and" orders.tsv
```

You should see this output:

```
chapter1 — bash — 80×24
robert:~/Desktop/analysis/chapter1$ grep "Chips and" orders.tsv
1       1       Chips and Fresh Tomato Salsa    NULL    $2.39
1       1       Chips and Tomatillo-Green Chili Salsa   NULL    $2.39
5       1       Chips and Guacamole     NULL    $4.45
7       1       Chips and Guacamole     NULL    $4.45
8       1       Chips and Tomatillo-Green Chili Salsa   NULL    $2.39
10      1       Chips and Guacamole     NULL    $4.45
13      1       Chips and Fresh Tomato Salsa    NULL    $2.39
15      1       Chips and Tomatillo-Green Chili Salsa   NULL    $2.39
18      1       Chips and Guacamole     NULL    $4.45
18      1       Chips and Tomatillo Green Chili Salsa   NULL    $2.95
20      1       Chips and Guacamole     NULL    $4.45
22      1       Chips and Guacamole     NULL    $3.99
25      1       Chips and Fresh Tomato Salsa    NULL    $2.39
28      1       Chips and Guacamole     NULL    $4.45
32      1       Chips and Guacamole     NULL    $3.99
33      1       Chips and Guacamole     NULL    $4.45
39      1       Chips and Fresh Tomato Salsa    NULL    $2.95
41      1       Chips and Guacamole     NULL    $4.45
42      1       Chips and Guacamole     NULL    $4.45
44      1       Chips and Guacamole     NULL    $4.45
45      1       Chips and Guacamole     NULL    $3.99
48      1       Chips and Guacamole     NULL    $4.45
49      1       Chips and Tomatillo Red Chili Salsa     NULL    $2.95
```

Here is one strategy for solving this problem: from this list of lines that correspond to orders of Chips with salsa, we extract only the part of the line that informs us about which salsa was obtained, and we count how many of each we observe.

Let's start by extracting only the part of each line that is relevant to us. In our case, notice that we are interested in column #3, the Item Name (verify that by doing `head "orders.tsv"`). To extract column #3 from our file, we can make use of a command called `cut` as follows:

```
grep "Chips and" orders.tsv | cut -f 3
```

Here, the **command-line option** `-f` specifies which field (or column) to extract, or cut out from the file. When you run that command, you should see that the output consist only of lines such as `Chips and Guacamole` or `Chips and Roasted Chili Corn Salsa`.

Now onto the last part. We would like to count how many of each Chips and Salsa was bought. However, this is a complex procedure and there isn't one command that can do all that (we'll have to use 2).

> **PRINCIPLE**: One of the UNIX philosophies that is true of Bash is that each command does only one thing but does it well.

```
```

Counting how many of each

Here we need the command `uniq -c` to count (hence the `-c`) how many of each lines of output it encounters. However, `uniq -c` also requires the input to be sorted. So the first thing we'll do here is to sort the list of Chips and Salsa items. We can do this very easily with a command that is conveniently called `sort`:

```
grep "Chips and" orders.tsv | cut -f3 | sort
```

Notice that, as a result of our list being sorted, all the lines with salsa of the same category are right next to each other. Next, we'll use `uniq -c` to essentially "condense" neighboring lines that are the same and in the process, count how many of each are seen:

```
grep "Chips and" orders.tsv | cut -f3 | sort | uniq -c
```

You should see the following output:

```
 110 Chips and Fresh Tomato Salsa
 479 Chips and Guacamole
   1 Chips and Mild Fresh Tomato Salsa
  22 Chips and Roasted Chili Corn Salsa
  18 Chips and Roasted Chili-Corn Salsa
  43 Chips and Tomatillo Green Chili Salsa
  48 Chips and Tomatillo Red Chili Salsa
  31 Chips and Tomatillo-Green Chili Salsa
  20 Chips and Tomatillo-Red Chili Salsa
```

We have a listing of how many of each kind of Salsa were bought with Chips, and it's clear that the vast majority of consumers (according to this dataset) preferred Guacamole with their chips. Upon closer examination, however, notice that "Roasted Chili Corn Salsa" and "Roasted Chili-Corn Salsa" were considered as different entries simply because of a 1 character difference, most likely due to error in data entry.

This brings us to an important point about data science. Often times, we're dealing with imperfect datasets where errors creep in. Unfortunately, unless we tell the computer to expect errors, it has no way of knowing–and no reason to expect–such typos. Had we taken the earlier approach of searching for each kind of salsa using `grep`, we would have missed this issue and it could have affected our final conclusions.

With that, congratulations on completing your first adventure!