LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community

FEBRUARY 2016 | ISSUE 262 | www.linuxjournal.com

A LOOK AT KDE's KStars Astronomy Program

Programming How-Tos

Program a BeagleBone Black to Help Brew Beer

Write a Short Script to Solve a Math Puzzle

+ Working with Command Arguments in Your Shell Scripts

Interview: Katerina Barone-Adesi on Developing the Snabb Switch Network Toolkit

WATCH: ISSUE OVERVIEW

Command-Line Tutorial:

Does Every Year Have a Friday the 13th?

Can you write a one-liner that answers this question for a given year? Can you write a short script around your one-liner and solve this puzzle for all time?

SOL LEDERMAN

Here's a fun little math problem that a handful of Linux commands, a bit of thought and a short shell script will help solve: does every year have a Friday the 13th? `cal`, `cut`, `grep`, `sed` and `cksum` will be your friends in this exploration.

Before engaging the computer, let's consider how you might solve this problem without one; you could examine some number of calendars. If you discover a year's calendar with no Friday the 13th, you're done. Otherwise, you eventually will come to believe that every year does have a Friday the 13th. "Eventually" is a long time away though. So, how many calendars do you need to look at? You may want to pause to consider this question before reading further. And, to support your exploration, you may want to use the Linux `cal` command to display calendars to your terminal.

Welcome back! Did you count 14 different calendars that you need to consider? Did you count seven? Both answers are correct, depending on your approach. (If you don't believe that looking at seven calendars is enough, see the suggested explorations at the end of the article.) Let's also consider leap years for this exploration. And, there's another consideration: is it enough to inspect the calendars for 14 consecutive years? Ponder that for a bit.

Let's dive in. How many unique calendars are there? Two calendars are "unique" if they look different—that is, if some months start on different days when you compare the two. Two calendars are the same if you could hang either one on the wall, and except for the year printed at the top of the calendar, you couldn't tell the difference. 2014 started on a Wednesday, and 2015 started on a Thursday. Those are two unique calendars. 2006 and 2012 both started on a Sunday. Are they the same calendar? Could you swap one for the other? No, because 2012 was a leap year, but 2006 wasn't. So, although January of both calendars was the same for those years, the other months weren't.

Considering that some years are leap years and that others aren't leads to one approach to counting calendars. Every year starts on one of the seven days. If you don't consider leap years (and there is an approach where you don't need to consider them), then there are seven different calendars. If you also consider leap years, you have seven more calendars to consider. So, if you examine 14 unique calendars, you will have the answer. You either will find a year that doesn't have a Friday the 13th or you won't.

Here is the part of the analysis where the computer is going to help. You need to look at 14 unique calendars. Will it be enough to look at the calendars for 2000 through 2013? That's a 14-year span. It turns out that that span does not include 14 unique calendars. In particular, 2001 and 2007 are the same calendar. Rather than trying to think through how many calendars you need to look at, let's take a different approach. Let's write a shell script that will review calendars until it has looked at 14 unique ones.

How can you make the computer tell you if calendars for two years are the same? Let's take a detour away from that question for a short while and consider this command: `cksum`. `cksum` displays the checksum of a file (or of stdin). What's a checksum? A checksum is a number associated with a file that is commonly used to detect transmission errors.

Imagine that you send a file from one computer to another via a noisy phone line. You want some assurance that the file wasn't corrupted during the transmission. What could you do? You could compare the sizes of the files on both ends. If the sizes are different, you would know the file copy is corrupt. But, if the sizes are the same, you certainly can't be confident that the content of the files is the same. You could transmit the file twice and compare the two copies byte by byte, but that would double the transmission time.

Checksums solve the problem of verifying that two files are identical without incurring a large computational or transmission cost.

Checksums solve the problem of verifying that two files are identical without incurring a large computational or transmission cost. The Linux `cksum` utility computes a single number—the checksum, for a file. The process for using the checksum is this: compute the checksum, transmit the file, compute the checksum for the received file and compare the checksums. If the checksums are identical, there is a near-100% probability that the two

files are identical. Near-100% is quite good enough for most purposes.

Now, let's return to the calendar problem. You want to find 14 unique calendars and check each to see if they have a Friday the 13th. Here's some pseudo-code that illustrates this approach:

```
Year = 2000          # Year increases until we
                     # have seen 14 unique
                     # calendars
ChecksumCount = 0    # This counts how many unique
                     # calendars we've seen
While ChecksumCount < 14 {
     Compute the checksum for Year
     If we have not seen this checksum yet {
          Add this checksum to the list of
             checksums we've seen
          Add 1 to ChecksumCount
     }
     Add 1 to Year
}
```

Why do you use checksums to compare calendars? Why don't you just create a file for each unique calendar and compare each calendar to every saved file? You could do that. But, once you work through the checksum approach, I think it will become clear that comparing files, with the `diff` command or some other approach, is clunkier. You decide.

It's time to dive in to some shell commands. I mentioned earlier that 2001 and 2007 are the same calendar. Compare the output of these two commands:

```
cal 2001
cal 2007
```

Now, compute the checksum of each calendar by piping the output of `cal` to `cksum`:

```
cal 2001 | cksum
3673415557 2014

cal 2007 | cksum
2655244645 2014
```

Hmmm. The two checksums are not the same. Why is that? (Note that "2014" is the number of characters in the output of `cal`. Ignore that and just look at the first number.) It's the title in the output of `cal` (the year) that makes the two calendars appear to be different. All you need to do is remove the first line, and you'll be good. `sed` is one of a number of Linux tools that easily can remove the first line of the output:

```
cal 2001 | sed '1d' | cksum
2408573533 1980
cal 2007 | sed '1d' | cksum
2408573533 1980
```

Note that `cksum` may generate a different checksum for your `cal` output. That's because `cal` has slightly different layouts on some flavors of Linux. As long as you're comparing `cal` output on the same machine, everything will work correctly.

`cksum` has confirmed that 2001 and 2007 are the same calendar. The next task is to build a list of the unique checksums that you've seen so that you can compare new calendar checksums to those on the list. Let's use a string to hold that list and initialize it to be empty:

```
checksums=""
```

Then, add the checksum for 2001 to the list:

```
checksums="$checksums 2408573533"
```

In order to add just the checksum to the list and not the number of characters in the output of `cal`, you will need to keep just the first part of `cksum`'s output and discard the second part. There are many ways to do this in bash. Here's one way:

```
cal 2001 | sed '1d' | cksum | awk '{print $1}'
2408573533
```

Awk is a very powerful text and data processing Linux tool. If you've never used awk, it's worth reading a tutorial about it and exploring it. For the purposes of this article, all you're doing with awk is telling it to print the first (whitespace-separated) field of the input ($1) from the pipe.

With the new checksum, 2408573533, use `grep` to search the list of checksums you've seen for this new one:

```
echo "$checksums" | grep -w 2408573533
```

Note that it's a good idea to use the `-w` flag with `grep` to match only 2408573533 as a word, with spaces, tabs and punctuation separating words. You don't, for example, want to match 24085735331234567 89.

You also will want to use `grep` with the `-q` option when you write your shell script to count Friday the 13ths. The option tells `grep` to search quietly and not to display any output. You then will use the return status from `grep` to know whether it matched anything:

```
echo $checksums | grep $newChecksum
if [ $? -eq 0 ] # if return value is 0
then
   # we matched something
fi
```

Whenever you see a unique

calendar, increment a counter:

```
numUniqueCalendars=$((numUniqueCalendars+1))
```

`$((..))` is a way you can do simple arithmetic in bash. Note that inside the `$((..))`, you don't put dollar signs in front of variable names.

When the counter reaches 14, you've seen all unique calendars.

Now that you have a general idea about how to use checksums, strings and `grep` to determine whether a calendar is unique and to count unique calendars, there is one more significant task. Given a calendar, how do you know if it has a Friday the 13th? Let's look at the output from `cal` for just one month:

```
cal 4 2001
     April 2001
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

April 13, 2001, is a Friday. It's easy enough for us humans to scan the Friday column looking for 13. How might a computer do the same? If you're not familiar with the Linux `cut` command, take a look at its man page and play with it a little before reading further.

`cut` allows you to select one or more columns. Sunday dates are in columns 1 and 2; column 3 is blank; Monday dates are in columns 4 and 5; and, if you keep counting columns, you see that Friday dates are in columns 16 and 17. Pipe the output of `cal` to `cut`, and you get all the Friday dates:

```
cal 4 2001 | cut -c16-17
Fr
 6
13
20
27
```

All you need now is to `grep` these Friday dates for the number 13:

```
cal 4 2001 | cut -c16-17 | grep 13
13
```

So, you now have a Linux pipeline that will tell you if a given year and month have a Friday the 13th. How can you generalize this approach to scan an entire year? Note that `cal` displays entire years in a 3x4 layout of three months across and four months down. You can use `cut` again and have it display three columns of Fridays. Note that you will need to

include a column with a space, either before or after the dates, otherwise the column values will run together:

```
cal 2001 | cut -c16-18,38-40,60-62
```

Note that the column ranges may be different in your version of Linux. I had to change column numbers to get the output to be correct on one of my Linux machines. Here is some of the output:

```
Fr Fr Fr
 6  4  1
13 11  8
20 18 15
27 25 22
      29


Fr Fr Fr
 6  3
13 10  7
20 17 14
27 24 21
   31 28
```

Notice that there are two Friday the 13ths in 2001. Let's add `grep` to the pipeline to filter out just the lines with 13 in them and use `wc` to count those lines:

```
cal 2001 | cut -c16-18,38-40,60-62 | grep -o 13 | wc -l
2
```

You may be wondering why you need the `-o` option to `grep`. Consider this:

```
echo 12 13 14 13 | grep 13
12 13 14 13
```

The number 13 appears twice in the list that you `echo`, but `grep` is matching the entire line, and when you pipe the output of `grep` to `wc -l` to count matches, you get this:

```
echo 12 13 14 13 | grep 13 | wc -l
       1
```

It would be nice to know how many Friday the 13ths there are in a year. This is where the `-o` option is useful:

```
echo 12 13 14 13 | grep -o 13
13
13
```

Adding `-o` tells `grep` to print only the matching part of lines and to print matches on multiple lines. Now you easily can count Friday the 13ths for any year. How many Friday the 13ths were there in 2014? Let's see:

```
cal 2014 | cut -c16-18,38-40,60-62 | grep -o 13 | wc -l
1
```

Congratulations! You've laid all of

the groundwork and should now be able to write a shell script to answer this article's burning question!

Let's summarize the programming approach used here. This should look very similar to the pseudo-code earlier in this article with one addition: you are going to count the number of Friday the 13ths in each unique calendar.

1. Pick a starting year. Does it matter what that year is?

2. For this year, compute its calendar checksum.

3. If the checksum is not on the unique checksum list, then you are looking at a calendar that you've not looked at previously. Add its checksum to your unique checksum list, and add one to the number of unique calendars you've seen. Count the number of times Friday the 13th appears in this year.

4. If you've seen 14 unique calendars, you're done. If not, add one to the year and loop back to step 2.

Here is a bash script that puts all of these ideas together. Note that the lines ending in \ need to not have a space, tab or any other character after them.

```
#!/bin/bash
# Starting with the year 2000, we
# compute how many times
# Friday the 13th occurs in every
# year. We stop when we have
# looked at 14 different calendars.
year=1999
checksums=""
numUniqueCalendars=0
echo "Year     Number of Friday the 13ths"
while [ $numUniqueCalendars -lt 14 ]
do
   year=$((year+1))
   sum=$(cal ${year} | sed '1d' | cksum |\
      awk '{print $1}')
   echo "$checksums" | grep -q $sum
   if [ $? -ne 0 ]
   then # It's a new calendar
      checksums="$checksums $sum"
      numUniqueCalendars=$((numUniqueCalendars+1))
      echo -n "${year} "
      cal ${year} | cut -c16-18,38-40,60-62 |\
         grep -o 13 | wc -l
   fi
done
```

Here's the output:

```
Year     Number of Friday the 13ths
2000        1
2001        2
2002        2
```

```
2003        1
2004        2
2005        1
2006        2
2008        1
2009        3
2010        1
2012        3
2016        1
2020        2
2024        2
```

Notice that you had to examine 25 consecutive calendars (between 2000 and 2024) to find 14 unique ones. And, notice...drum roll... Friday the 13th occurs once, twice or three times in every year.

I chose to start checking calendars starting in 2000. What would have happened if I had chosen some other year?

I hope you appreciate the power of a handful of Linux commands, some logical thinking and a short shell script to solve a fun math challenge. If you enjoyed this problem, consider exploring these related problems:

1. How would using `ncal` instead of `cal`, if it's available on your system, simplify your code?

2. Does every year have a Monday the 13th? What's the most any year can have? For this exploration (and for #6), you'll need to generalize the concept of pulling out one or more columns from a calendar to be able to select any day of the week.

3. If you exclude January and February, does every year still have a Friday the 13th?

4. 2015 has three Friday the 13ths. When will that happen next?

5. How likely is the 13th of a month to fall on a Friday vs. the other days of the week? Have the computer scan every month for 100 years of calendars to find out.

6. Which day/date pairs occur every year? For example, does every year have a Monday the 31st? This exploration generalizes #2.■

Sol Lederman is a math geek and general techie. As a writer and programmer, he especially loves to communicate technical things to a broad audience. Read more about Sol at http://solleder man.com.

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.