Parsing Command-Line Options with getopt

DAVE TAYLOR

Make your shell scripts more flexible and more command-line-friendly by accepting command-line arguments/flags.

I've talked before about how I am a lazy shell script programmer. It might be because I'm simply not a full-time professional software developer, and I don't even administer my own servers anymore—I outsource the job to Wisconsin.

Regardless of how much I program nowadays though, I still find myself needing simple little applications—tiny programs that do one simple task well.

And, then there are the throwaway scripts that stick around, ultimately becoming a mainstay of one's toolkit, spreading out to cover multiple functions and mysteriously growing to 100 lines or more.

I have one of those in my toolkit, a script that originally was intended simply to figure out the dimensions of a graphic file and produce the proper height and width attributes for an HTML image tag.

Now the script scale.sh has grown to 133 lines and does a variety of different, albeit related tasks. No surprise, it's also grown to have a variety of command-line arguments, as shown here:

```
$ ./scale.sh

Usage: scale {args} factor [file or files]
  -a      use URL values for APparenting.com site
  -b      add 1px solid black border around image
  -i      use URL values for intuitive.com/blog site
  -k KW   add keywords KW to the ALT tags
  -r      use 'align=right' instead of <center>
  -s      produces succinct dimensional tags only

A factor 0.9 for 90% scaling, 0.75 for 75%, or max width in pixels.
A factor of '1' produces 100%.
```

Crack open the code, and you'll see my dirty little scripting secret—a very sloppy approach to parsing command-line options:

```
if [ "$1" = "-a" ] ; then
  baseurl="www.apparenting.com/Images/"; shift
fi
```

I did warn you that I was a lazy programmer, right? This is a pretty classic way to parse and process command-line arguments, actually. Check the value of $1, and if it's a known flag, change a default variable or two, then use the shift command to move $2 → $1, $3 → $2 and so on, effectively deleting the processed flag from the command-line args.

The problem is, when you have more than one or two flags, this really doesn't work. I step through the command flags alphabetically in my script—for example, invoking the script as `scale -r -a` will fail. It'll process the -r flag but never see the -a and generate an error condition.

Fortunately, there's a very nice Linux command called getopt that lets you parse through your command flags in a far more sophisticated manner.

getopt In Shell Scripts

The getopt command first requires that you let it rearrange how your command flags are organized, then you use the set command to update all the positional variables. After that, you can step through the positional variables with a case statement.

The first step is:

```
args=`getopt FLAGS $*`
set -- $args
```

where FLAGS should be the individual letters of known and accepted command flags. If a flag has an argument that goes with it (like -s 30), append a colon to it.

For my script, it looks like this:

```
args=`getopt abik:rs $*`
set -- $args
```

To see what happens, I've added a bonus echo statement. Here's the result:

```
$ scale -abs -k fdsf 100 *png
args = -a -b -s -k fdsf -- 100 blooeeh.png
```

As you can see, getopt separates out each and every command flag and adds a -- flag that indicates when the command flags end—simple, really!