Now that the args have been restructured, parsing is relatively easy, though it looks pretty complicated (warning, I've stripped out a few clauses for simplicity):

```
for i; do
  case "$i" in
    -a  ) baseurl="www.apparenting.com/Images/"
          shift ;;
    -k  ) keywords=" ($2)"
          shift ; shift  ;;
    -s  ) verbose=0
          shift ;;
    --  ) shift; break ;;
  esac
done
```

Let's read this backward. At the -- option, the loop will exit due to the break. Until that's hit, the for loop will just keep iterating, stepping through all the flags specified. This is how the order of the flags becomes irrelevant.

Each time a flag is matched, the desired action is taken, variables are set and so on, then the shift command shows up again to move all the command flags down one (for example, $2 to $1, $3 to $2 and so on).

Shell script case statement matching lines are all in the form of:

```
regex ) actions  ;;
```

The double semicolon is an oddity, but that's how you indicate the end of an individual case match, hence the notation shown above.

Grabbing the argument for the -k flag is easy too, because getopt has made sure that it's a separate argument, and since we're using shift as we go along to move things around, $2 will always be the argument itself.

Finally, also notice that as a stylistic approach, I have the double semicolon with a leading space. That's just so when I eyeball the script, I quickly can recognize if there are any cases that are missing the double semicolon.

The only piece missing is some error handling, because right now, if a bad flag is encountered, here's what happens:

```
$ scale -ax 100 *png
getopt: illegal option -- x
```

Nice, but the script doesn't catch the error condition or stop running—not so good.

To fix it, immediately after the call to getopt, simply test the return code:

```
if [ $? != 0 ] ; then ...
```

In the conditional, you probably would put a usage statement and an exit command. For my script, I actually also test to ensure that there are a minimum of two arguments on the command line as well, because the script is never valid without them:

```
if [ $? != 0 -o $# -lt 2 ] ; then
  echo ""
  echo "Usage: scale {args} factor [file or files]"
  echo ""

  ... stuff skipped ...

  exit 0
fi
```

At this point in our shell script writing journey, I certainly hope you can read that rather cryptic conditional statement and understand what it does.

Ultimately, it's a bit of work to parse command-line flags the right way, but it makes for a far more flexible and robust shell script.■

Dave Taylor has been involved with UNIX since he first logged in to the on-line network in 1980. That means that, yes, he's coming up to the 30-year mark now. You can find him just about everywhere on-line, but start here: www.DaveTaylorOnline.com.