COMP 2718: Shell Scripts: Part 3

By: Dr. Andrew Vardy

Outline

- The Compound Commands: [[ ]] and (( ))
- `until` Loops
- The `for` Loop: Original Form
- The `for` Loop: C Language Form

The Compound Commands: [[ ]] and (( ))

The `test` command has a more modern variant. Recall, the usual structure of `test` using square brackets. e.g.:

```
[ -e "$file" ]
```

The newer form is the compound command `[[ ]]` and has two new abilities. The first, has to do with regular expressions (to be covered soon). The second is that the == operator supports matching with wildcards:

```
$ FILE=foo.bar
$ if [[ $FILE == foo.* ]]; then
> echo "$FILE matches pattern 'foo.*'"
> fi
foo.bar matches pattern 'foo.*'
```

The compound command `(( ))` is used for **artithmetic truth tests**. Such a test has an exit status of 0 (success) whenever the embedded expression is non-zero. For example:

```
$ if ((1)); then echo "It is true."; fi
It is true.

$ if ((0)); then echo "It is true."; fi
$
```

`(( ))` is **designed for integers only**. This allows for the following features:

- We can go ahead and use the symbols <, >, and == (rather than `-lt`, `-gt`, and `-eq`)
- Variable names don't need to be prefixed with `$`

Consider the following example. . .

```
----------------------------------------------------
INT=$1
if ((INT == 0)); then
    echo "INT is zero."
else
    if ((INT < 0)); then
        echo "INT is negative."
    else
        echo "INT is positive."
    fi
    if (( ((INT % 2)) == 0)); then
        echo "INT is even."
    else
        echo "INT is odd."
    fi
fi
----------------------------------------------------
```

until Loops

The syntax of an `until` loop is much like a while loop:

```
until condition_commands; do commands; done
```

The choice of `while` vs. `until` is rather arbitrary—either can be used. Choose whichever seems to match the logic of the situation.

e.g. This script prints numbers in decimal, octal, and hex up to 16.

```
----------------------------------------
i=0
until [ "$i" -eq 17 ]; do
    printf "dec: %d, \toctal: %o, \thex: %X\n" $i $i $i
    i=$(($i + 1))
done
----------------------------------------
```

Notice the `printf` command. Most programming languages have an equivalent command that provides a controlled way of displaying text output. It is preferable to `echo` when you want fine-grained control.

Lets see how this script is simplified using the `(( ))` compound command. First the original:

```
---------------------------------------------------
i=0
until [ "$i" -eq 17 ]; do
    printf "dec: %d, \toctal: %o, \thex: %X\n" $i $i $i
    i=$(($i + 1))
done
---------------------------------------------------
```

Now using `(( ))`:

```
---------------------------------------------------
i=0
until ((i == 17)); do
    printf "dec: %d, \toctal: %o, \thex: %X\n" $i $i $i
    ((i=i+1))
done
---------------------------------------------------
```

The for Loop: Original Form

The original bash for loop has the following form:

```
for variable [in words]; do
    commands
done
```

The variable is typically set to each of the values in the list of items denoted words. For each of these items, variable has that value and commands is executed.

We can illustrate a basic `for` loop right on the command-line:

```
$ for i in A B C D; do echo $i; done

A
B
C
D
```

`for` loops work really well with various types of expansions. For example, pathname expansion (globbing):

```
$ for i in distros*.txt; do echo $i; done

distros-by-date.txt
distros-dates.txt
distros-key-names.txt
```

Brace expansion:

```
$ for i in {A..C}; do echo $i; done

A
B
C
```

The following shows the use of command substitution to generate the list of words that the `for` loop operates on. The `strings` command separates the file into readable words.

```
----------------------------------------------------------
if [[ -r $1 ]]; then
    max_word=
    max_len=0
    for j in $(strings $1); do
        len=$(echo $j | wc -c)
        if (( len > max_len )); then
            max_len=$len
            max_word=$j
        fi
    done
    echo "DONE"
    echo "'$max_word' ($max_len characters)"
fi
----------------------------------------------------------
```

The for Loop: C Language Form

bash supports a second type of for loop that borrows its syntax from C (also C++ and Java):

```
for (( expression1; expression2; expression3 )); do
    commands
done
```

The expressions (1, 2, and 3) are arithmetic expressions. This form is equivalent to the following:

```
(( expression1 ))
while (( expression2 )); do
    commands
    (( expression3 ))
done
```

Here is an example of the C-style `for` loop:

```
-----------------------------------------------------
for (( i=0; i<5; i=i+1 )); do
     echo $i
done
-----------------------------------------------------
```

```
0
1
2
3
4
```