**Math in Shell Scripts**

Shell script variables are by default treated as strings, **not numbers**, which adds some complexity to doing math in shell script. To keep with script programming paradigm and allow for better math support, languages such Perl or Python would be better suited when math is desired. However, it is possible to do math with shell script. In fact, over the years, multiple facilities have been added to Unix to support working with numbers.

Note

As we will see, some of the commands used to facilitate math are a little picky about things like spaces around operators.

**declare**

You may recall, that when the text book introduced the **declare** statement, it said that it is not always needed. So what do you get by declaring a variable to be an integer? The following example illustrates that a declared integer is not treated as a string.

$ n=6/3

$ echo $n

6/3

$ declare -i n

$ n=6/3

$ echo $n

2

When you do not need the **declare** statement is when you will use a program or built-in command to evaluate a math statement.

**expr**

An old Unix program that can evaluate math is **expr**. **expr** became popular in the days of the Bourne Shell, which did not support math. With Bash and Korn shell, it is generally not needed. Since it is a command, command substitution is needed. We are still treating the variable as a string. As you can see, it is picky about spaces.

$ z=5

$ z=`expr $z+1` ---- Need spaces around + sign.

$ echo $z

5+1

$ z=`expr $z + 1`

$ echo $z

6

**let**

A Bash and Korn shell built-in command for math is **let**. As you can see, it is also a little picky about spaces, but it wants the opposite of what **expr** wanted. **let** also relaxes the normal rule of needing a $ in front of variables to be read.

$ let z=5

$ echo $z

5

$ let z=$z+1

$ echo $z

6

$ let z=$z + 1 # --- Spaces around + sign are bad with let

-bash: let: +: syntax error: operand expected (error token is "+")

$let z=z+1 # --- look Mom, no $ to read a variable.

$echo $z

7

An alternate form of let is to wrap the whole statement in double parenthesis. This form is more forgiving about spaces.

$ ((e=5))

$ echo $e

5

$ (( e = e + 3 ))

$ echo $e

8

$ (( e=e+4 )) # -- spaces or no spaces, it doesn't matter

$ echo $e

12

**bc**

What if you want to do math with floating point numbers or you have some fairly complicated math to do? Neither form of let, supports floating point numbers. The bc command is needed. But you have to treat the variables as strings.

Here is what happens when we try to do floating point math with the shell:

$let r=3.5

-bash: let: r=3.5: syntax error in expression (error token is ".5")

$(( r = 3.5 ))

-bash: ((: r = 3.5 : syntax error in expression (error token is ".5 ")

bc

An arbitrary precision calculator language. **bc** may either be run interactively, or as a shell script command. In interactive mode, type cntrl-d (EOF) to exit.

SYNOPSIS

**bc**

**bc** EXPRESSION

Here are some examples:

$ bc

bc 1.06

Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.

This is free software with ABSOLUTELY NO WARRANTY.

For details type `warranty'.

3 + 2

5

obase=2

12

1100

<cntrl-d>

Remember to type cntrl-d (EOF) to exit from interactive mode.

$r=3.5

$s=`echo "$r + 2.2" | bc`

$echo $s

5.7

$ z = `echo $z + 1 | bc`

-bash: z: command not found

# -- spaces around = sign are bad

(shell thing, not bc)

$ z=`echo "$z + 1" | bc`

$ echo $z

8

$ z=`echo "$z+1" | bc` -- spaces don't matter with bc

$ echo $z

9

**Numeric Boolean expressions:**

Note that the arithmetic tests shown in Figure 6-27, page 223, must be used with the (( )) form of **let**. Otherwise, according to the **test** man page, the following numeric tests are used.

INTEGER1 -eq INTEGER2

INTEGER1 is equal to INTEGER2

INTEGER1 -ge INTEGER2

INTEGER1 is greater than or equal to INTEGER2

INTEGER1 -gt INTEGER2

INTEGER1 is greater than INTEGER2

INTEGER1 -le INTEGER2

INTEGER1 is less than or equal to INTEGER2

INTEGER1 -lt INTEGER2

INTEGER1 is less than INTEGER2

INTEGER1 -ne INTEGER2

INTEGER1 is not equal to INTEGER2

That is to say, the following two if statements are identical:

if (( x < y )); then

statements

fi

if [ $x -lt $y ]; then

statements

fi

And for logical expressions using floating point math, the **bc** command returns 1 for logical true expressions and 0 for false expressions, thus testing for result of 1 achieves a logical expression:

if [ $( echo "$t < 3.4" | bc ) -eq 1 ]; then

statements

fi