**eval reads arguments as input to the shell (the resulting commands are executed).**

**This allows using the value of a variable as a variable.**

paul@deb503:~/test42$ answer=42

paul@deb503:~/test42$ word=answer

paul@deb503:~/test42$ eval x=\$$word ; echo $x

**Result:** 42

**Both in bash and Korn the arguments can be quoted.**

kahlan@solexp11$ answer=42

kahlan@solexp11$ word=answer

kahlan@solexp11$ eval "y=\$$word" ; echo $y

42

**. (( ))**

**The (( )) allows for evaluation of numerical expressions.**

paul@deb503:~/test42$ (( 42 > 33 )) && echo true || echo false

true

paul@deb503:~/test42$ (( 42 > 1201 )) && echo true || echo false

false

paul@deb503:~/test42$ var42=42

paul@deb503:~/test42$ (( 42 == var42 )) && echo true || echo false

true

paul@deb503:~/test42$ (( 42 == $var42 )) && echo true || echo false

true

paul@deb503:~/test42$ var42=33

paul@deb503:~/test42$ (( 42 == var42 )) && echo true || echo false

false

**The let built-in shell function instructs the shell to perform an evaluation of arithmetic**

**expressions.**

**It will return 0 unless the last arithmetic expression evaluates to 0.**

[paul@RHEL4b ~]$ let x="3 + 4" ; echo $x

7

[paul@RHEL4b ~]$ let x="10 + 100/10" ; echo $x

20

[paul@RHEL4b ~]$ let x="10-2+100/10" ; echo $x

18

[paul@RHEL4b ~]$ let x="10\*2+100/10" ; echo $x

30

**There is a difference between assigning a variable directly, or using let to evaluate**

**the arithmetic expressions (even if it is just assigning a value).**

**You can sometimes simplify nested if statements with a case construct.**

**[paul@RHEL4b ~]$ ./help**

**What animal did you see ? lion**

**You better start running fast!**

**[paul@RHEL4b ~]$ ./help**

**What animal did you see ? dog**

**Don't worry, give it a cookie.**

**[paul@RHEL4b ~]$ cat help**

kahlan@solexp11$ let dec=15 ; let oct=017 ; let hex=0x0f

kahlan@solexp11$ echo $dec $oct $hex

15 15 15

kahlan@solexp11$ dec=15 ; oct=017 ; hex=0x0f

kahlan@solexp11$ echo $dec $oct $hex

15 017 0x0f