Linux Shell Scripting

Part-2

I. Arrays, Operators, and Control Flows

I.1. Arrays

An array is defined as the collection of elements. Usually, array contains same type of values in programming languages. On the other hand, array in shell scripting is a variable which can contain both same or different type of multiple values. Indexing starts with zero in an array. Bash doesn't support multi-dimensional array. We can declare arrays in different ways:

- Indirect declaration

  ```
  arr[0]=John
  ```

- Explicit declaration: Declare the array before assigning values.

  ```
  declare -a arr
  ```

- Compound assignment: Declare array with a set of values.

  ```
  names=(John Andrew David Michael Richard)
  ```

Some of the examples on how to access array values:

```
echo ${names[*]}       #prints all the elements
echo ${names[@]}       #prints all the elements
echo ${names[0]}       #prints 0th element i.e., John
echo ${#names[*]}      #prints length of array i.e., 5
echo ${#names}         #prints length of 0th element i.e., 4
echo ${names[1]:3}     #${arr[element]:start_index} i.e., rew
echo ${names[4]:2:4}   #${arr[element]:start_index:count} i.e., char
echo ${names[*]/ch/y}  #look for ch, replace with y. Prints John Andrew David Miyael Riyard
unset names[1]         #removes 1st element i.e., Andrew
```

I.2. Math Operators and Expressions

In bash, arithmetic operations can be performed using variables. However, we cannot use variables directly for operations as shell consider variables as strings by default. To solve this problem, there are some external programs such as `expr`, `let`, and `(( ))` or we can say commands which can be used to perform arithmetic operations. Following are some of the arithmetic operators to perform math operations:

- +, -, *, / : add, subtract, multiply, divide

- % : modulus (remainder)
- =, ==, != : Assignment, equality, inequality
- ++ : increment
- - - : decrement

1. `let`: Used to evaluate arithmetic expressions. It saves the result to the variable.

```
let v=5+2
let v+=2
let "b=1+2"
let a++
let b=100*$a
let c=100/3
let d=a+1
let a=6 b=4 c=a*b d=c/2
```

2. `expr`: It prints the answer. This is similar to `let` but does not save the answer to a variable.

```
expr 1 - 1
expr 1-1
expr "1 - 1"
v=$(expr 2 + 2)
expr $v * 2
expr $v \* 2
echo "c= `expr 5 + $v`"
expr "a=$((2**3))"
```

3. `(( ))`: We can use double parantheses to perform arithmetic operations in which expression is enclosed by parantheses.

```
((a+=2))
echo $((b+2))
echo $(( 4/3 ))
c=$((a+2))
((a=3)) && echo $a
echo $((a=6,b=4,c=a*b,d=c/2))
echo $((a=6,b=4,c=a*b,d=c/2)) && echo $a $b $c $d
echo $((a=6,b=4,c=a*b,d=c/2)) ; echo $a $b $c $d
```

I.3. Floating Point Arithmetic

Bash doesn't allow floating-point arithmetic. We can use power of 10 to get the float ouput.

1. `printf`: A shell built-in which can be used to get the floating number.

```
printf %.3f "$((10**3 * 2/3))e-3"
printf %.2f $((10**3 *  4/3 ))e-3
```

2. awk: We can perform floating-point arithmetic with awk command. It can also be used for pattern scanning and processing in files or documents.

```
awk "BEGIN {print 100/3}"
awk "BEGIN {print -100- -2}"
awk '{print}' hello.txt
awk '/ab/{print}' hello.txt
awk '{print $2}' hello.txt
awk '{print NR,$1}' hello.txt
awk 'NR==3 {print NR,$0}' hello.txt
awk 'NR==3,NR==5 {print NR,$0}' hello.txt #prints 3rd to 5th rows with all the columns
```

3. bc: It provides the basic mathematical calculator applicaton through command line. It can be used to perform arithmetic calculations.

```
echo "15.6+299.33*2.3/7.4" | bc
bc <<< "15.6+299.33*2.3/7.4"  #output=108.6
echo "scale=2; 2/3" | bc      #output=.66
echo $(( 10#2 + 2#10 ))       #output=4
echo "obase=2; 5"  |bc        #output=101
```

4. test: It is a bash built-in command that evaluates a conditional expression to be either true or false.

```
test 27 -gt 2 && echo "Yes"
test 10 -gt 20 && echo "Yes"
test 15 -ne 10 && echo Yes || echo No
```

I.4. Conditional Statement: If and else

Conditional statements help you perform decision-making. We can perform tasks with multiple conditions. For this purpose, we can use conditions with if, if else, if elif and/or nested if statements.

```
#!/bin/bash
echo -n "Enter a number: "
read VAR
if [[ $VAR -lt 10 ]]
then
  echo "The variable is less than 10."
fi
```

I.5. Case Statement

In case of multiple conditions with a set or array of elements, it is not always the best solution to use if-else statements. `case` statement enhance the readability of the code and makes it easier to handle multiple choices.

```
#!/bin/bash
echo -n "Enter name:"
read name
echo -n "The language is "
case $name in
    Rohit)
        echo -n "Hindi"
        echo
        ;;
    Rahul | Aman)
        echo -n "English"
        echo
        ;;
    Anirban | Sam)
        echo -n "Hindi and English"
        echo
        ;;
    *)
        echo -n "Unknown"
        echo
        ;;
esac
```

I.6. Loops

Loops allow us to execute a block of code repeatedly through iteration.

- Count-controlled Loop: Definite execution. Executes a block of code for a fix number of times. We can fix the number of iterations with a step length using `for` loop.
- Condition-controlled Loop: Executes a block of code until a condition is true. It generally uses `while` loop.
- Infinite Loop: Repeats the loop infinitely. This can be called endless or forever loop. Infinite loop can use `while true`.
- Collection-controlled Loop: Repeats the loop for each element in the set or array. This can be performed using `for in` loop.

1. For loop:

```
#!/bin/bash
d=(ubuntu mac windows redhat)
for os in ${d[*]}        #collection controlled
do
```

```
    echo "I love ${os}"
done
for ((x=0 ; x<5 ; x++)) #count controlled
do
    echo "x equals to $x"
done
```

2. While loop:

```
#!/bin/bash
x=5
while (( $x>0 ))
do
    echo $(( x-- ))
done
while :
do
    echo "Infinite loop"
done
```

3. Until loop: This is opposite of While loop.

```
#!/bin/bash
x=5
until (( $x<=0 ))
do
    echo $(( x-- ))
done
until false
do
    echo "Infinite loop"
done
```

I.7. Interrupt Loops

We can interrupt a loop using keywords like `break` and `continue` if we want to skip the remaining lines of code in the loop and either come out of the loop or continue the loop for the next iteration. Both the commands can be used in any loop.

- Break: `break` is used to come out of the loop. It can take a parameter `n` which allows you to exit from n^{th} level of enclosing loop.

```
#!/bin/bash
for ((x=0 ; x<5 ; x++))
do
    echo "x equals to $x"
    if [[ $x -eq  3 ]]
    then
        break
    fi
```

```
done

for (( x=0 ; x<5 ; x++ ))
do
    for (( y=0 ; y<2 ; y++ ))
    do
        if [[ $x -eq  3 ]]
        then
            break 2
        fi
        echo "x is $x and y is $y"
    done
done
```

- Continue: It is similar to `break` command except that it immediately runs the next iteration of the loop and exits only the current iteration, rather than the entire loop.

```
#!/bin/bash
num=(1 2 3 4 5 6 7 8 9 10)
echo ${num[*]}
for n in ${num[*]}
do
   m=`expr $n % 2`
   if [[ $m -eq 0 ]]
   then
      echo "Number $n is an even number"
      continue
   fi
   echo "Number $n is an odd number"
done
```