UNIX

Lecture 9 : SHELL programming (bash) (Part 3)

Filipe Vasconcelos[1]
[1]ESME, Lille, filipe.vasconcelo@esme.fr

ESME
PARIS•BORDEAUX•LILLE•LYON

Outline

1. Arrays
2. Functions

Intended Learning Outcomes

ILO4

Produce, test, and verify the results of Shell scripts (in Bash) to perform tasks ranging from simple to complex algorithms.

Outline

Arrays in Bash

`bash` has two types of arrays

- **Regular arrays**: Elements are indexed by integers.
- **Associative arrays**: Elements are indexed by strings.

Regular Arrays

Declaration:

```
array=("value0" "value1" "value2")
```

To access an element of the array:

```
${array[2]} #Note:  the first index of the array is 0
```

To define or modify an element of an array:

```
array[3]="new value"
```

The entire content of the array:

```
echo "${array[*]}" #Note:  no preservation of string integrity
echo "${array[@]}" #Note:  preservation of string integrity
```

To retrieve the total number of elements in the array:

Arrays

Example (on GitHub: `example_array.sh`)

```
#!/bin/bash
array=("Monday" "Tuesday" "Wednesday")
echo "Initialization"
echo "First value: ${array[0]}"
echo "Second value: ${array[1]}"
echo "Third value: ${array[2]}"

echo
echo "Displaying all elements"
echo "${array[*]}"

echo
echo "Displaying the number of elements"
echo "${#array[@]}"

array[0]="Friday"
echo
echo "Modification"
#Exercise: display the values with a for loop
```

Iterating Through an Array

```
array=("one two" three)

for element in "${array[@]}"
do
    echo ${element}
done

for element in ${array[@]}
do
    echo ${element}
done
```

```
one two
three



one
two
three
```

Be cautious, as some array traversal syntax can lead to confusion, especially when using `${array[*]}`. The best approach is to test your script.

Other Operations on Classic Arrays

List of all defined indices

```
echo "${!array[@]}"
```

Copying an array

```
array_copy=( "${array[@]}" )
```

Adding elements to an array

```
array+=("four" "five")     # at the end
array=("zero" "${array[@]}") # at the beginning
```

Deleting an array

```
unset array
```

Associative Arrays

Declaration is done with the built-in command `declare`:

```
declare -A assoc_array=([fr]=Paris [it]=Rome [pt]=Lisbon)
```

Some Operations

List all keys of an associative array:

```
echo "${!assoc_array[@]}"
```

To iterate through an associative array:

```
for key in "${!assoc_array[@]}"
do
    echo ${assoc_array[$key]}
done
```

Adding an element:

Some Exercises

Exercise 1

Write a shell program ‘card‘ that displays the name of a randomly drawn card from a deck of 32 cards. You will use two arrays: an array ‘suits‘ and an array ‘values‘. You will use the ‘RANDOM‘ environment variable.

Exercise 2

Write a shell program ‘deal‘ that creates a set of 5 different cards drawn randomly from a deck of 32 cards and displays the contents of the set.

Outline

Functions

- A function is a set of instructions that allows you to perform multiple tasks with different input parameters.
- Functions make your Bash program more readable and structured, facilitating program development.
- Once a function is defined, you can call it as many times as needed within your script.
- A function must be declared before it is used.
- Recursive calls are possible.

Functions

Declaration:

```
function foo {
    # Example function
    echo "inside the foo function"
}
```

or

```
foo() {
    echo "inside the foo function"
}
```

Calling the function:

```
# Simple function call
foo
```

Functions

- Functions can take arguments passed to them and return an exit code to the script using the ‘return‘ statement.
- You access the value returned by the predefined variable $?.
- The function refers to arguments passed to it by their position (as if they were positional parameters), i.e., $1, $2, and so on.

Example:

```
max() {
    if [ "$1" -ge "$2" ]
    then
        return $1
    else
        return $2
    fi
}
read -p "Enter two numbers: " n1 n2
max ${n1} ${n2}
echo "The maximum of ${n1} and ${n2} is $?
```

Functions
Global Variable and Local Variable

- A global variable is defined outside of the function. It is visible throughout the script.
- A local variable is defined inside the function. It is only visible within the function.

Example:

```
c=0
function square
{
    a=$1
    c=$(($a*$a))
    return
}
square 9
echo "The square of 9 is ${c}"
```

Very Common Use

Displaying help for the user

```
#!/bin/bash

usage() {
    echo "Usage: $0 [a] [n]"
    echo "a: a string"
    echo "n: an integer"
    exit 1
}

[ $# -ne 2 ] && usage

...
<rest of the script>
...
exit 0
```

Small Exercise

Exercise 1

Write a shell program `fact` that calls either the iterative or recursive version to calculate the factorial of a number passed as a parameter to the script. You will write two Bash functions, `factiter` and `factrecur`. The script `fact` will accept 2 arguments: the first is a string defining the version to use, and the second is the number for which you want to calculate the factorial.