Seneca | SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

Unix Scripting

Lecturer: Shahdad Shariatmadari

July 2020

Agenda

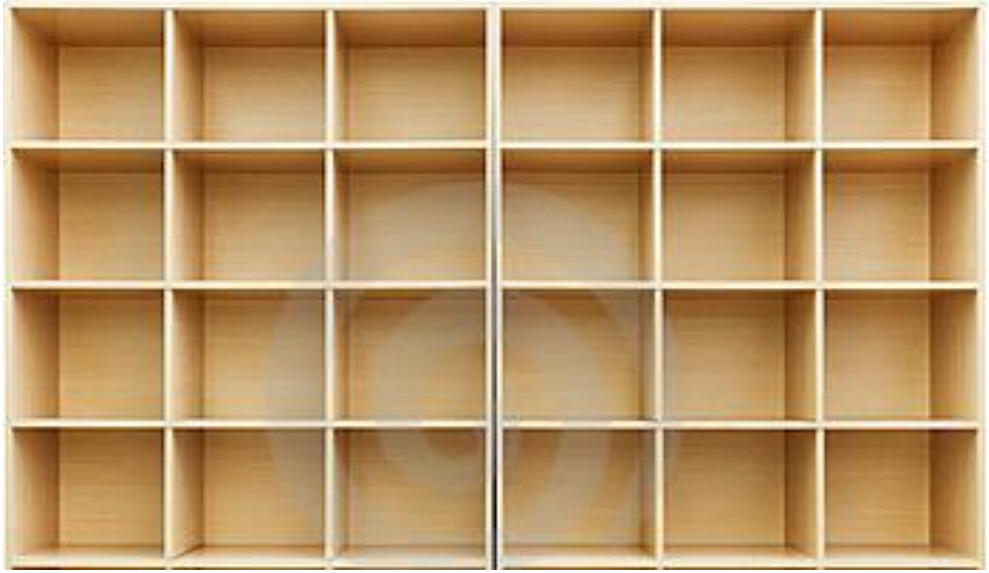
- Array in Shell Scripting
  - How to declare
  - How to process

Introduction to Arrays

- A **variable** is a memory location that can store a value. It can be thought of as a box in which values are stored. The value held in the box can change, or vary. But each variable can only hold one item of data.
- An array is a series of memory locations – or 'boxes' – each of which holds a single item of data, but with each box sharing the same name. All data in an array must be of the same **data type**.

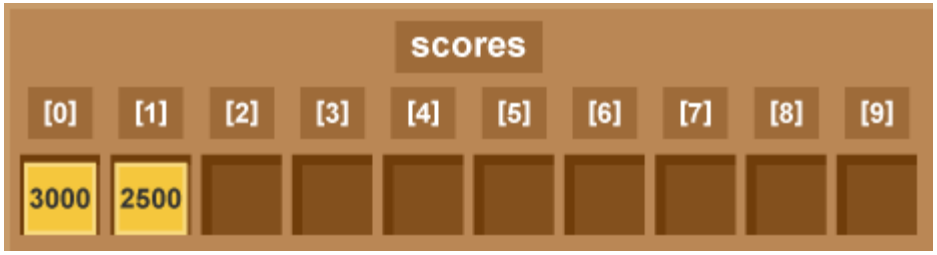
Array

- Arrays are like shelves



Array

- For example, imagine that a score table in a game needs to record ten scores.
  - Var Score1, Score2, …, Score10
- Instead of having ten variables, each holding a score, there could be one array that holds all the related data:
  - score(10)



Array

- An array is a variable containing multiple values.
- Using the following syntax to declare an array:
  - **ARRAY[INDEX]=value**
- Explicit declaration of an array is done using the **declare** built-in:
  - **declare -a ARRAYNAME**
- Array variables may also be created using compound assignments in this format:
  - **ARRAY=(value1 value2 ... valueN)**

How to access to an array element

- In order to refer to the content of an item in an array, use curly braces.
- **ARRAY=*(one two three)***
- To display first array element
  - ***echo ${ARRAY[0]}***
  - Array indexing always start with 0.
- To display all array elements
  - **echo ${ARRAY[*]}**

Read Array from Input

- We can also read/assign values to array during the execution time using the *read* shell-builtin.
- `read -a array`
- Upon executing the above statement inside a script, it waits for some input. We need to provide the array elements separated by space (and not carriage return)

Various Operations on Arrays

- array=(apple bat cat dog elephant frog)
- #print first element
- `echo ${array[0]}`
- `echo ${array:0}`

Various Operations on Arrays

- array=(apple bat cat dog elephant frog)
- #display all elements
- `echo ${array[@]}`

Various Operations on Arrays

- `array=( apple bat cat dog elephant frog )`
- What does the following command do?
- `echo ${#array[0]}`
  - #length of first element
- `echo ${#array}`

Various Operations on Arrays

- array=(apple bat cat dog elephant frog)
- `echo ${array[@]:1}`
  - #display all elements except first one
- `echo ${array[@]:1:4}`
  - #display elements in a range

Various Operations on Arrays

- array=(apple bat cat dog elephant frog)
- `echo ${array[@]/a/A}`
  - #replacing substring

Shell Parameter Expanssion

- The basic form of parameter expansion is ${*parameter*} which we used it in array dereferencing
- The basic form of parameter substitution is {parameter/pattern/string}

- ${*parameter*:*offset*:*length*}
  - This is referred to as Substring Expansion. It expands to up to *length* characters of the value of *parameter* starting at the character specified by *offset*.

Traverse Array Elements using Loop

- To traverse through the array elements we can also use for loop.

```
for i in "${array[@]}"
   do
   #access each element as $i. . .
   done
```

Activity 1

- Create an script which
  - Ask user to enter 6 city_name and store them into array
  - Display array elements in reverse order in the output

Command Substitution with Arrays

- Command substitution assigns the output of a command or multiple commands into another context.
- Here in this context of arrays we can insert the output of commands as individual elements of arrays. Syntax is as follows.
  - `array=( $(command) )`
- Try this:
  - `array=($(date))`
  - `echo ${array[@]}`

Activity 2

- Develop an script which
  - Create an array with the following data
  - `I spring like spring on spring in spring`
  - Replace the spring with something else (whatever you want)
  - Display the array in the output

Activity 3

- Create a text file on the matrix, name it “mydata.txt”
  - Enter 15 people name in the file (in each line)
- Develop a new script which read the file and store it into array
  - `read -a people <<<$(cat mydata.txt)`
  - `Or : people=$(cat mydata.txt)`
  - Add a loop to display array in reverse order

Associative array

- The array that can store string value as an index or key is called associative array.
- An associative array can be declared in bash by using the **declare** keyword and the array elements can be initialized at the time of array declaration or after declaring the array variable.
  - `declare -A assocArray1`
    `assocArray1[fruit]=Mango`
    `assocArray1[bird]=Cockatail`
    `assocArray1[flower]=Rose`
    `assocArray1[animal]=Tiger`

Accessing the Associative Array

- Array elements of an associative array can be accessed individually or by using any loop.
- echo ${assocArray1[bird]}
- for key in "${!assocArray1[@]}"

```
do
    echo $key
done
echo "${!assocArray2[@]}"
```

Example

- `declare -A assocArray2=`

  `( [HDD]=Samsung [Monitor]=Dell [Key board]=A4Tech )`

Activity4

- Create a script which:
  - Create an associative array and add 5 different pairs of username/password
  - Display all array elements (usernames+passwords) on the output