Bash Array

In this topic, we will demonstrate the basics of bash array and how they are used in bash shell scripting.

An array can be defined as a collection of similar type of elements. Unlike most of the programming languages, arrays in bash scripting need not be the collection of similar elements. Since Bash does not discriminate the string from a number, an array may contain both strings and numbers.

Bash does not provide support for the multidimensional arrays; we cannot have the elements which are arrays in themself. Bash provides support for one-dimensional numerically indexed arrays as well as associative arrays. To access the numerically indexed array from the last, we can use negative indices. The index of '-1' will be considered as a reference for the last element. We can use several elements in an array.

Bash Array Declaration

Arrays in Bash can be declared in the following ways:

Creating Numerically Indexed Arrays

We can use any variable as an indexed array without declaring it.

To explicitly declare a variable as a Bash Array, use the keyword 'declare' and the syntax can be defined as:

```
1. declare -a ARRAY_NAME
```

where,

ARRAY_NAME indicates the name that we would assign to the array.

Note: Rules of naming a variable in Bash are the same for naming an array.

A general method to create an indexed array can be defined in the following form:

```
ARRAY_NAME[index_1]=value_1
ARRAY_NAME[index_2]=value_2
```

```
ARRAY_NAME[index_n]=value_n
```

where keyword 'index' is used to define positive integers.

Creating Associative Arrays

Unlike numerically indexed arrays, the associative arrays are firstly declared. We can use the keyword 'declare' and the -A (uppercase) option to declare the associative arrays. The syntax can be defined as:

```
declare -A ARRAY_NAME
```

A general method to create an associative array can be defined in the following form:

```
declare -A ARRAY_NAME
ARRAY_NAME[index_foo]=value_foo
ARRAY_NAME[index_bar]=value_bar
ARRAY_NAME[index_xyz]=value_xyz
```

where index_ is used to define any string.

We can also write the above form in the following way:

```
declare -A ARRAY_NAME

ARRAY_NAME=(
    [index_foo]=value_foo
    [index_bar]=value_bar
    [index_xyz]=value_xyz
)
```

Bash Array Initialization

To initialize a Bash Array, we can use assignment operator (=), by specifying the list of the elements within parentheses, separated by spaces like below:

```
ARRAY_NAME=(element_1st element_2nd element_Nth)
```

Note: Here, the first element will have an index 0. Also, there should be no space around the assignment operator (=).

Access Elements of Bash Array

To access the elements of a Bash Array, we can use the following syntax:

```
1. echo ${ARRAY_NAME[2]}
```

Print Bash Array

We can use the keyword 'declare' with a '-p' option to print all the elements of a Bash Array with all the indexes and details. The syntax to print the Bash Array can be defined as:

```
1. declare -p ARRAY_NAME
```

Array Operations

Once an array is assigned, we can perform some useful operations on it. We can display its keys and values as well as modify it by appending or removing the elements:

Reference Elements

To reference a single element, we are required to know the index number of the element. We can reference or print any element using the following syntax:

```
1. ${ARRAY_NAME[index]}
```

Note: Curly braces ${} are required to avoid shell's filename expansion operators.

For example, let's print an element of an array with an index of 2:

Bash Script

```
#!/bin/bash
#Script to print an element of an array with an index of 2

#declaring the array
declare -a example_array=( "Welcome""To""Javatpoint" )

#printing the element with index of 2
echo ${example_array[2]}
```

Output

Javatpoint

If we use @ or * in the place of a specified index, it will expand to all members of the array. To print all the elements, we can use the following form:

Bash Script

```
#!/bin/bash
#Script to print all the elements of the array

#declaring the array
declare -a example_array=( "Welcome""To""Javatpoint" )

#Printing all the elements
echo "${example_array[@]}"
```

Output

```
Welcome to Javatpoint
```

The only difference between using @ and * is that the form is surrounded with double quotes while using @. In the first case (while using @), the expansion provides a result in a word for each element of the array. It can be better described with the help of "for loop". Assume we have an array with three elements, "Welcome", "To" and "Javatpoint":

```
1. $ example_array= (Welcome to Javatpoint)
```

Applying a loop with @:

```
1. for i in "${example_array[@]}"; do echo "$i"; done
```

It will produce the following result:

```
Welcome
To
Javatpoint
```

Applying a loop with *,a single result will be produced holding all the elements of the array as a single word:

```
Welcome To Javatpoint
```

It is important to understand the usage of @ and * because it is useful while using the form to iterate through array elements.

Printing the Keys of an Array

We can also retrieve and print the keys used in indexed or associative arrays, instead of their respective values. It can be performed by adding the ! operator before the array name as below:

```
1. ${!ARRAY_NAME[index]}
```

Example

```
#!/bin/bash
#Script to print the keys of the array

#Declaring the Array
declare -a example_array=( "Welcome""To""Javatpoint" )

#Printing the Keys
echo "${!example_array[@]}"
```

Output

```
0 1 2
```

Finding Array Length

We can count the number of elements contained in the array by using the following form:

```
1. ${#ARRAY_NAME[@]}
```

Example

```
#!/bin/bash

#Declaring the Array
declare -a example_array=( "Welcome""To""Javatpoint" )

#Printing Array Length
echo "The array contains ${#example_array[@]} elements"
```

Output

```
The array contains 3 elements
```

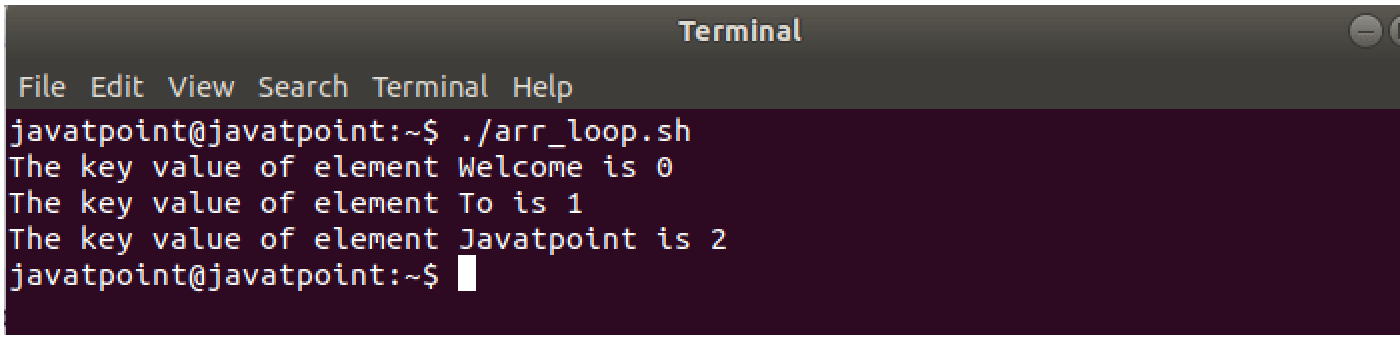
Loop through the Array

The general method to iterate over each item in an array is by using the 'for loop'.

Example

```
#!/bin/bash
#Script to print all keys and values using loop through the array

declare -a example_array=( "Welcome""To""Javatpoint" )

#Array Loop
for i in "${!example_array[@]}"
do
echo The key value of element "${example_array[$i]}" is "$i"
done
```

Output



```
javatpoint@javatpoint:~$ ./arr_loop.sh
The key value of element Welcome is 0
The key value of element To is 1
The key value of element Javatpoint is 2
javatpoint@javatpoint:~$
```

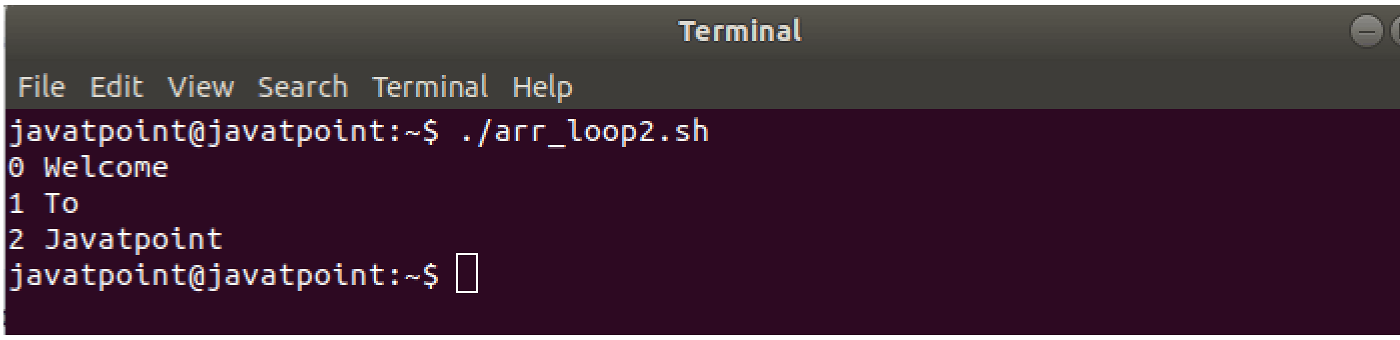
Another common method to loop through an array is to retrieve the length of the array and use the C-style loop:

Bash Script

```
#!/bin/bash
#Script to loop through an array in C-style

declare -a example_array=( "Welcome""To""Javatpoint" )

#Length of the Array
length=${#example_array[@]}

#Array Loop
for (( i=0; i < ${length}; i++ ))
```

```
do
echo $i ${example_array[$i]}
done
```

Output



```
javatpoint@javatpoint:~$ ./arr_loop2.sh
0 Welcome
1 To
2 Javatpoint
javatpoint@javatpoint:~$ 
```

Adding Elements to an Array

We have an option to add elements to an indexed or associative array by specifying their index or associative key respectively. To add the new element to an array in bash, we can use the following form:

1. ARRAY_NAME[index_n]="New Element"

Example

```
#!/bin/bash

#Declaring an array
declare -a example_array=( "Java""Python""PHP""HTML" )

#Adding new element
example_array[4]="JavaScript"

#Printing all the elements
echo "${example_array[@]}"
```

Output

Java Python PHP HTML JavaScript

Another method for adding a new element to an array is by using the += operator. There is no need to specify the index in this method. We can add one or multiple elements in the array by using the following way:

Example

```
#!/bin/bash

#Declaring the Array
declare -a example_array=( "Java""Python""PHP" )

#Adding new elements
example_array+=( JavaScript CSS SQL )

#Printing all the elements
echo "${example_array[@]}"
```

Output

```
Java Python PHP JavaScript CSS SQL
```

Updating Array Element

We can update the array element by assigning a new value to the existing array by its index value. Let's change the array element at index 4 with an element 'Javatpoint'.

Example

```
#!/bin/bash
#Script to update array element

#Declaring the array
declare -a example_array=( "We""welcome""you""on""SSSIT" )

#Updating the Array Element
example_array[4]=Javatpoint

#Printig all the elements of the Array
echo ${example_array[@]}
```

Output

```
We welcome you on Javatpoint
```

Deleting an Element from an Array

If we want to delete the element from the array, we have to know its index or key in case of an associative array. An element can be removed by using the '**unset**' command:

```
1. unset ARRAY_NAME[index]
```

An example is shown below to provide you a better understanding of this concept:

Example

```
#!/bin/bash
#Script to delete the element from the array

#Declaring the array
declare -a example_array=( "Java""Python""HTML""CSS""JavaScript" )

#Removing the element
unset example_array[1]

#Printing all the elements after deletion
echo "${example_array[@]}"
```

Output

```
Java HTML CSS JavaScript
```

Here, we have created a simple array consisting of five elements, "Java", "Python", "HTML", "CSS" and "JavaScript". Then we removed the element "Python" from the array by using "unset" and referencing the index of it. The index of element "Python" was '1', since bash arrays start from 0. If we check the indexes of the array after removing the element, we can see that the index for the removed element is missing. We can check the indexes by adding the following command into the script:

```
1. echo ${!example_array[@]}
```

The output will look like:

```
0 2 3 4
```

This concept also works for the associative arrays.

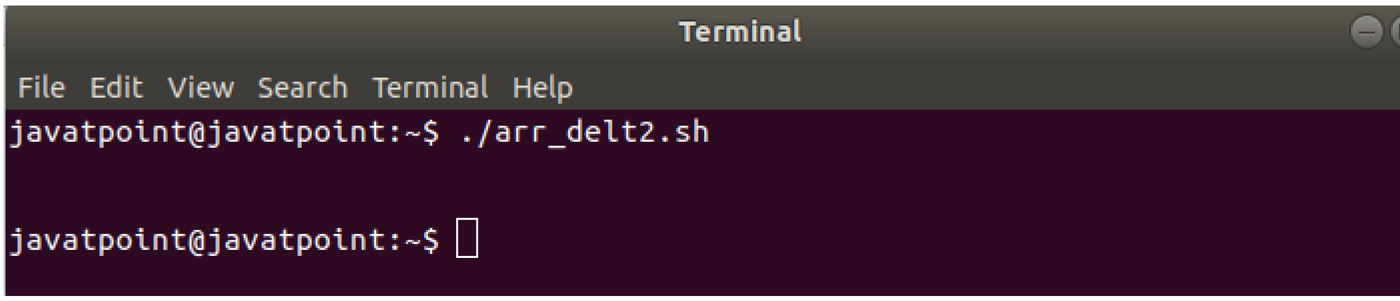
Deleting the Entire Array

Deleting an entire array is a very simple task. It can be performed by passing the array name as an argument to the '**unset**' command without specifying the index or key.

Example

```
#!/bin/bash
#Script to delete the entire Array

#Declaring the Array
declare -a example_array=( "Java""Python""HTML""CSS""JavaScript" )

#Deleting Entire Array
unset example_array

#Printing the Array Elements
echo ${!example_array[@]}

#Printing the keys
echo ${!example_array[@]}
```

Output



```
javatpoint@javatpoint:~$ ./arr_delt2.sh

javatpoint@javatpoint:~$ 
```

There will be no output if we try to print the content of the above script. An empty result is returned because the array doesn't exist anymore.

Slice Array Elements

Bash arrays can also be sliced from a given starting index to the ending index.

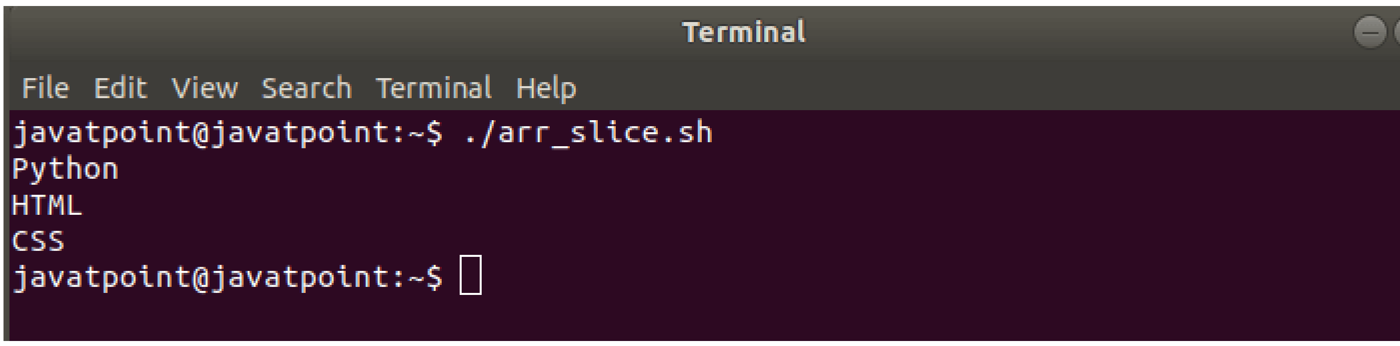
To slice an array from starting index 'm' to an ending index 'n', we can use the following syntax:

1. SLICED_ARRAY=(${ARRAY_NAME[@]:m:n}")

Example

```
#!/bin/bash
#Script to slice Array Element from index 1 to index 3

#Declaring the Array
example_array=( "Java""Python""HTML""CSS""JavaScript" )

#Slicing the Array
sliced_array=("${example_array[@]:1:3}")

#Applying for loop to iterate over each element in Array
for i in "${sliced_array[@]}"
do
echo $i
done
```

Output



```
javatpoint@javatpoint:~$ ./arr_slice.sh
Python
HTML
CSS
javatpoint@javatpoint:~$
```