BASH Arrays

SCRIPTING ESSENTIALS

DR. BURKMAN

Making an array

We declare an array like this:

```
declare -a my_array_name=(1 2 3 a b c "the dog")
```

Each item is only separated by a space. If an item has spaces then enclosed it in double quotes.

You can also start an array by assigning a value to an index number. This works without using declare:

```
my_other_array_name[0]="dog"
```

And you can declare an empty array:

- declare -a my_array=()

You can clear an array with:

- my_array=()

Adding elements to an array

You can add elements to an array like this:

You can also:

- myArray[0]=“dog”

Empty an array:

- myArray=()

Copy an array:

- Copyarr=(${origarr[*]})

```
myArray+=(2)
myArray+=(a b c)
echo ${myArray[*]}

pet="dog"
myArray+=($pet)
echo ${myArray[*]}

myArray[3]=${myArray[0]}
echo ${myArray[*]}

array1+=(1 2 3)
array2+=(a b c)
array3+=(${array1[*]} ${array2[*]})
echo ${array3[*]}
```

Array output

We get the output of all array elements like this: `echo ${my_array_name[*]}`

- Note: you can use either * or @
- Pay attention to that syntax. Dollar sign, brace, array name, bracket, index, bracket, brace

To get the zeroth element: `echo ${my_array_name[0]}`

To get a slice: `echo ${my_array_name[*]:2:3}`

- This would get a slice starting at index 2, and returning 3 items. Either the 2 or the 3 could, instead, be a variable.

Leaving off the second number returns the rest of the array from that starting index:
`echo ${my_array_name[*]:2}`

Array output

You can address a single item with a negative index value: `echo ${my_array_name[-1]}`

Getting a slice with negative numbers is tricky, though: `echo ${my_array_name[*]: -3}`

- You *must* put a space between the colon and the minus sign. This example will return the last three items in the array, in their normal left to right order.
-

Removing items and indexes

You can remove an item from an index with unset: `unset myArray[0]`

- This doesn't remove the indexed element (the box). If you echo the array it will skip over this index because it's value is null. But you could call myArray[0] because it's still an indexed element.

To actually remove an element you have to copy all but that element back to the array:

```
jim=2
myArray=(${myArray[*]:0:$jim} ${myArray[*]:$(($jim + 1))})
```

- Note that there is a space before the second $.
- This is just adding two arrays. The first goes from 0 to the chosen index. The second goes from the chosen index +1 to the end.

Array length

You can get the length of an array or element by putting # in front of the array.

- This will return the length of the whole array `echo ${#myArray[*]}`
- This will return the length of the item at index 3 `echo ${#myArray[3]}`

Iterating through an array, method #1

This will iterate through the items and print each item out:

```
for i in ${my_array_name[*]};
        do
                echo $i
        done
```

I would use the method only for printing out the elements of an array. For matching array elements with some variable method 2 is better.

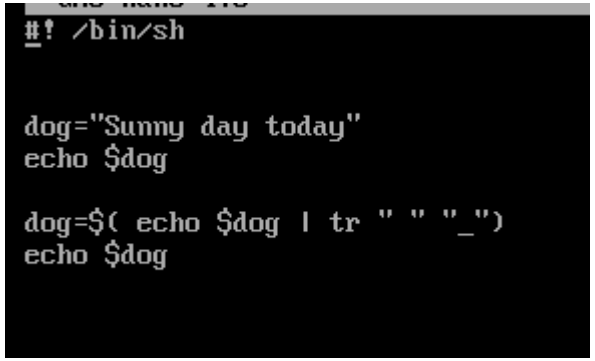
Iterating through an array, method #2

Putting ! in front of the array returns the index numbers rather than the element values. In this example I could be checking to see if a given text string is in an array. If so, it would echo the index number. Overall, this method is more stable and unless just printing the elements of an array I would use this method.

```
value="rat“
for i in ${!myArray[*]};
do
        if [[ ${myArray[$i]} == $value ]]; then
                echo $i
        fi
done
```

Using Translate (tr)

Arrays don't deal nicely with items that have spaces. This is why you'll see older output with underscores, like Jim_Burkman. We can pipe a variable through tr and replace spaces with underscores like this:



```
#! /bin/sh


dog="Sunny day today"
echo $dog

dog=$( echo $dog | tr " " "_")
echo $dog
```

That line reads like "have dog take the value of the result of (print the stuff in dog and send that (pipe it) to tr. Replace each space with an underscore."

- tr "thing to replace" "thing you are replacing it with"

Using Sort with an array

Sort will sorts lines of text and/or numbers. To use this with an array we will have to iterate through the array elements then pipe them to sort. And, of course, assign all this back to our array:

```
#! /bin/sh

clear
declare -a myArray=(1 5 3 -1 a H z A L M r 12)
echo ${myArray[*]}

myArray=($(
        for i in ${myArray[*]};
        do
        echo $i
        done | sort))
echo ${myArray[*]}
```

sort -r for reverse sort, -n to sort numerically, -nr reverse numeric, -u remove duplicates

https://www.geeksforgeeks.org/sort-command-linuxunix-examples/

Functions

We will set up a function like this:

```
function_name () {
  commands
}
```

And we call it be just using the name

```
hello_func () {
  echo "Hello World"
}
```

Functions

We can pass the function values and they will be assigned to the variable $1, $2, etc. in the order they are passed.

```
#!/bin/sh

hello_func () {
        echo "Hello $1 $2"
}

hello_func "Jim" "Nita"
```

Resources

https://tecadmin.net/working-with-array-bash-script/

https://www.thegeekstuff.com/2010/06/bash-array-tutorial/

https://www.unix.com/shell-programming-and-scripting/180613-sorting-output-echo.html

https://stackoverflow.com/questions/15028567/get-the-index-of-a-value-in-a-bash-array

https://linuxize.com/post/how-to-compare-strings-in-bash/

https://linuxize.com/post/bash-functions/