Bash Shell Scripting (part 6)

Purpose of Arrays
Using Arrays
Working with Temporary Files

Bash Shell Scripting (part 6)

Purpose of Arrays
Using Arrays
Working with Temporary Files

Purpose of Arrays

We finish shell scripting by using arrays by reading and storing networking information for a Linux server (in this case, your c7host).

Arrays are an incredibly useful tool that works like a "storage container" of variables. Arrays have names like variables, but are numbered for quick storage and access of related elements. Indices are I.D. numbers for array access. The first index starts at 0 (i.e. zero) instead of 1.

They are very useful when used with loops to store or display data or information within variables. Also, common functions (e.g. sorting) can be used with arrays.

How Arrays Work

Unlike with other programming languages (like C, C++), you do not have to define an array.

You can store data into array elements in a number of ways.
Index numbers must start from zero.

Examples:

```
array_name[0]="I Like OPS235" # store array elements individually
array_name[1]="I am on lab6"

array_name=(I  Like  OPS235)  # stores 3 array elements as same time
```

How Arrays Work

You can display the arrays a number of different ways:

Individually:

```
echo ${array_name[0]}
echo ${array_name[1]}
etc...
```

Display **all** Elements in the Array:

```
echo ${array_name[*]}
echo ${array_name[@]}
```

NOTE: Usually loops (such as the for loop) are very useful for inputting data into array elements as well as displaying array elements. The next slide provides examples of using a loop to enter and display array elements.

How Arrays Work

Example:

```
# Use for loop to store array elements
for((x=0; x<5; x++)) # Example of a regular array
do
  read -p "Enter item #$((x+1)): " item[$x]
done

# Use for loop to display stored array elements
 for((y=0; y<5; y++))
do
  echo ${item[y]}
done
```

Place the code above into a shell script, set execute permissions and run the shell script to see how it works.

Purpose of Associative Arrays

As opposed to using index numbers to access or store data into an array element, you can also use what is referred to an Associative Array.

Associative arrays use key-words instead of numbers. Therefore the array elements are associated with words, names, etc that are easier to remember as opposed to numbers.

How Associative Arrays Work

You would need to define an Associative Array. This is like saying that you are going to use a storage container to store your data (access by keywords instead of numbers).

```
declare -A array_name
```

Once you have set your array, you can now start to store data into array elements. Numbers (referred to as indices) within square brackets following the array name can contain data. Index numbers must start from zero.

Examples:

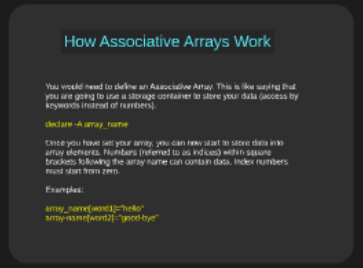
```
array_name[word1]="hello"
array-name[word2]="good-bye"
```

How Associative Arrays Work

Example:

```
set a b c # Example of an "Associative Array"
for x
do
  read -p "Enter item $x: " item["$x"]
  echo item[$x] is: ${item["$x"]}
done
```

Place the code above into a shell script, set execute permissions and run the shell script to see how it works.



Using Temporary Files

When creating temporary files, it is important NOT to store on a user's account (to avoid overwriting their existing files). Instead, temporary files can be created in the /tmp directory.

The $$ variable can be used as the filename extension which assigns the current PID of the shell script running to make the filename unique, and allow easy removal at the end of the shell script by deleting ALL files in the /tmp directory with the extension: .$$

Example:

```
ls -lR > /tmp/temp-file.$$
grep secret /tmp/temp-file.$$
rm /tmp/*.$$
```

Bash Shell Scripting (part 6)

Purpose of Arrays
Using Arrays
Working with Temporary Files