Bash Scripting: Arrays and Functions

Array

- Bash provides one-dimensional array variables
- Assign values to array:

```
array=( one two three )
files=( "/etc/passwd" "/etc/group" "/etc/hosts" )
limits=( 10  20  26  39  48)
```

- Access array element : ${array_name[index]}
  - indexed using integers and are zero-based.

  ${array[1]}
- To access all items in arary: ${array_name[*]} , ${array_name[@]}
- To access array length: len=${#x[@]}

To Iterate Through Array Values

```
#!/bin/bash
# declare an array called array and define 3 vales
 array=( one two three )
for i in "${array[@]}"
do
    echo $i
done
```

Bash function

- Functions: to increase modularity and readability
  - More efficient than breaking scripts into many smaller ones
- Syntax to define a function:

```
function functionname()
{
    commands . .
}
```

  - **function** is a keyword which is optional.
  - **functionname** is the name of the function
    - No need to specify argument in ( )
  - **commands** – List of commands to be executed I
    - **exit status** of the function is exit status of last command executed in the function body.

Function call

- Call bash function from command line or script
  - $ functionname arg1 arg2
  - When shell interprets a command line, it first looks into the special built-in functions like break, continue, eval, exec etc., then it looks for shell functions.
- function defined in a shell start up file (e.g.,.bash_profile).
  - available for you from command line every time you log on

About functions

- **Parameter passing**: $1, $2, …
- **Result returning**
  - Use echo command
  - Through setting a variable
  - return command: to return an exit status

```
#! /bin/bash
calsum(){
        echo `expr $1 + $2`
}
x=1;y=2;
sum=`calsum $x $y`
```

```
calsum(){
        sum=`expr $1 + $2`
}
x=1;y=2;
calsum $x $y
echo z=$sum
```