bash Functions and Arrays

Functions

- More efficient than scripts as they are kept in memory when the script is read in
- Help in organizing long scripts into manageable modules
- Defined by one of the two methods (no functional difference between the two forms):
  1. Method 1
     ```bash
     function fn
     {
       shell commands
     }
     ```
  2. Method 2
     ```bash
     fn()
     {
       shell commands
     }
     ```

```bash
#!/bin/bash

hello()
{
  if [ $# -eq 0 ]
  then
    echo "Hello World"
    return
  fi

  echo "Hello $*"
  return
}

hello
hello John
```

- The functions in the environment can be seen in alphabetical order) by the command `declare -f`
- A function can be deleted by `unset -f fn`
- Positional parameters in functions work exactly as they do in shellscripts
  - Positional parameters are local to the functions
  - `$0` stays the same inside a function because functions execute in the environment of shellscript
- Local variables
  - Variables defined inside a function are local to the function and override a variable with the same name in the calling script

Command Precedence
• Commands are executed in the following order of precedence
  1. Aliases
  2. Keywords (function) and other control statements
  3. Functions
  4. Built-in commands such as cd and type
  5. Scripts and executables, searched through PATH

• The exact version of command used can be found by type
  type ls
  type -all ls

Arrays in bash

• Any variable can be used as a 1D array
  – Identified as var[index]
  – bash has two types of arrays: indexed arrays and associative arrays

Indexed arrays

• Accessed by an index
  – Index starts at 0
  – Index can be any positive integer, up to 599147937791

• Values are assigned by
  var[index]=value

• Examples
  color[1]="red"
  color[2]="green"
  color[0]="blue"
  – Values need not be assigned in any specific order

• Another way to assign values, known as compound statement, is
  color=([2]=green [1]=red [0]=blue)

• If you specify the elements in order, you can specify them as
  color=(red blue green)

• The above values can be accessed by
  for i in 0 1 2
do
    print ${color[$i]}
done
Arrays and Functions

- If you specify an index at some point in compound assignment, the values get assigned in consecutive locations from that point on
  
  ```bash
  color=(red [2]=blue green)
  ```
  
  - This array has four elements, with index 1 element null
  - Reassigning to an existing array with a compound statement will lose existing values in array
  
  ```bash
  color=([5]=violet yellow)
  ```

- Index array need not be declared though they can be using the command `declare -a`

- You can also declare arrays of any size by
  
  ```bash
  declare -a color[3]
  ```
  
  with the elements indexed from 0 to 2
  
  - You can simply add more elements to it by assigning an element with a new index

- You can see all the values by using `*` or `@
  
  ```bash
  echo ${color[*]}
  ```
  
  - `*` expands the array to one string with values separated by first characters of `IFS`
  - `@` expands the array to separate words

- The indices that have been assigned can be listed by `!` as
  
  ```bash
  echo ${!color[@]}
  ```

- The number of elements, or the length of a specific element, can be found by `#`
  
  ```bash
  echo ${#color}
  echo ${#color[1]}
  ```

- List all the elements by
  
  ```bash
  for i in ${!color[@]}
  do
    echo ${color[$i]}
  done
  ```

**Associative arrays**

- The index can be any arbitrary string

- Associative arrays must be declared with `typeset -A`

- Examples
  
  ```bash
  typeset -A shade
  shade[apple]=red
  shade[banana]=yellow
  shade[grape]=purple
  ```
• The above values can be accessed by

```bash
for i in apple banana grape
do
  print ${shade[$i]}
done
```

• You can access all elements by using * for the index, for both indexed as well as associative arrays

• Other features of indexed arrays are available as well

```bash
for i in ${!shade[@]}
do
  printf "%-10s%-10s\n" $i ${shade[$i]}
done
```