UNIX / LINUX - SHELL FUNCTIONS

In this chapter, we will discuss in detail about the shell functions. Functions enable you to break down the overall functionality of a script into smaller, logical subsections, which can then be called upon to perform their individual tasks when needed.

Using functions to perform repetitive tasks is an excellent way to create **code reuse**. This is an important part of modern object-oriented programming principles.

Shell functions are similar to subroutines, procedures, and functions in other programming languages.

Creating Functions

To declare a function, simply use the following syntax –

```
function_name () {
   list of commands
}
```

The name of your function is **function_name**, and that's what you will use to call it from elsewhere in your scripts. The function name must be followed by parentheses, followed by a list of commands enclosed within braces.

Example

Following example shows the use of function –

Live Demo

```
#!/bin/sh

# Define your function here
Hello () {
   echo "Hello World"
}

# Invoke your function
Hello
```

Upon execution, you will receive the following output –

```
$./test.sh
Hello World
```

Pass Parameters to a Function

You can define a function that will accept parameters while calling the function. These parameters would be represented by **$1**, **$2** and so on.

Following is an example where we pass two parameters *Zara* and *Ali* and then we capture and print these parameters in the function.

Live Demo

```
#!/bin/sh

# Define your function here
Hello () {
   echo "Hello World $1 $2"
```

```
}

# Invoke your function
Hello Zara Ali
```

Upon execution, you will receive the following result –

```
$./test.sh
Hello World Zara Ali
```

Returning Values from Functions

If you execute an **exit** command from inside a function, its effect is not only to terminate execution of the function but also of the shell program that called the function.

If you instead want to just terminate execution of the function, then there is way to come out of a defined function.

Based on the situation you can return any value from your function using the **return** command whose syntax is as follows –

```
return code
```

Here **code** can be anything you choose here, but obviously you should choose something that is meaningful or useful in the context of your script as a whole.

Example

Following function returns a value 1 –

Live Demo

```
#!/bin/sh

# Define your function here
Hello () {
   echo "Hello World $1 $2"
   return 10
}

# Invoke your function
Hello Zara Ali

# Capture value returnd by last command
ret=$?

echo "Return value is $ret"
```

Upon execution, you will receive the following result –

```
$./test.sh
Hello World Zara Ali
Return value is 10
```

Nested Functions

One of the more interesting features of functions is that they can call themselves and also other functions. A function that calls itself is known as a ***recursive function***.

Following example demonstrates nesting of two functions –

Live Demo

```
#!/bin/sh
```

```
# Calling one function from another
number_one () {
   echo "This is the first function speaking..."
   number_two
}

number_two () {
   echo "This is now the second function speaking..."
}

# Calling function one.
number_one
```

Upon execution, you will receive the following result –

```
This is the first function speaking...
This is now the second function speaking...
```

Function Call from Prompt

You can put definitions for commonly used functions inside your ***.profile***. These definitions will be available whenever you log in and you can use them at the command prompt.

Alternatively, you can group the definitions in a file, say ***test.sh***, and then execute the file in the current shell by typing –

```
$. test.sh
```

This has the effect of causing functions defined inside ***test.sh*** to be read and defined to the current shell as follows –

```
$ number_one
This is the first function speaking...
This is now the second function speaking...
$
```

To remove the definition of a function from the shell, use the unset command with the **.f** option. This command is also used to remove the definition of a variable to the shell.

```
$unset .f function_name
```