Table Of Contents

nixCraft: Linux Tips, Hacks, Tutorials, And Ideas In Blog Format
http://www.cyberciti.biz/

Bash Shell Script Function Examples

Posted by Vivek Gite <vivek@nixcraft.com>

[1]

How do I create a shell script function using Bash under UNIX / Linux operating systems?

Functions are nothing but small subroutines or subscripts within a Bash shell script. You need touse to break up a complex script into separate tasks. This improves overall script readability and ease of use. However, shell function cannot return value. They return a status code.

Declare Shell Function

All functions must be declared before they can be used. The syntax is:

```
function name {
  Commands
}
```

OR

```
name () {
 Commands
 return $TRUE
}
```

You can call function by typing its name:

```
name
```

Example

Create a shell script called file.sh:

```
#!/bin/bash
# file.sh: a sample shell script to demonstrate the concept of Bash shell functions
# define usage function
usage () {
 echo "Usage: $0 filename"
 exit 1
}

# define is_file_exits function
# $f -> store argument passed to the script
is_file_exits () {
 local f="$1"
 [[ -f "$f" ]] && return 0 || return 1
}
# invoke  usage
# call usage() function if filename not supplied
[[ $# -eq 0 ]] && usage

# Invoke is_file_exits
if ( is_file_exits "$1" )
then
 echo "File found"
```

```
else
 echo "File not found"
fi
```

Run it as follows:

```
chmod +x file.sh
./file.sh
./file.sh /etc/resolv.conf
```

Task: Export functions

You need to use export command:

```
fname(){
  echo "Foo"
}

export -f fname
```

Task: Make readonly functions

You create functions at the top of the script and set the readonly attribute with the readonly command:

```
fname(){
  echo "Foo"
}

usage(){
  echo "Usage: $0 foo bar"
  exit 1
}

readonly -f usage
readonly -f fname
```

Task: Local variables functions

Use the local command to create local variables:

```
#!/bin/bash
# gloabal x and y
x=200
y=100

math(){
  # local variable x and y with passed args
  local x=$1
  local y=$2
  echo $(( $x + $y ))
}

echo "x: $x and y: $y"
# call function

echo "Calling math() with x: $x and y: $y"
math 5 10

# x and y are not modified by math()
echo "x: $x and y: $y after calling math()"
echo $(( $x + $y ))
```

Task: Recursion

A recursive function call itself. Recursion is a useful technique for simplifying some complex algorithms, and breaking down complex problems.

```
#!/bin/bash
foo(){
  # do something
  # if not false call foo
  foo
}

# call foo
foo
```

See Recursive function [2] for more details.

Recommend readings:

- **Chapter 9: Functions [3] from the Linux shell scripting wiki**

URLs in this post:

[1] Image: **http://www.cyberciti.biz/faq/category/bash-shell/**
[2] Recursive function: **http://bash.cyberciti.biz/guide/Recursive_function**
[3] Chapter 9: Functions: **http://bash.cyberciti.biz/guide/Writing_your_first_shell_function**