

Array

Palani Karthikeyan

abpalanikarthik@gmail.com

Array

- An array is a variable containing multiple values of same type or may be of different types.
- No maximum limit to the size of an array.
- No requirement that member variables be indexed or assigned contiguously.
- Array index starts with zero.

1.Declaring An Array And Assigning Values

- In bash , array is created automatically when a variable is used in the following format:

Syntax: name[index]=value

- name is any name for an array.
- index could be any number or expression that must evaluate to a number greater than or equal to zero.
- You can declare an explicit array using declare -a array name.
- To access an element from an array use curly brackets like **\${name[index]}**.

Example 1

```
$ cat array1.sh
```

```
#!/bin/bash  
os[0]=unix  
os[1]="linux program"  
os[2]=minix  
echo ${os[0]}  
echo ${os[*]}
```

Output of the above script:

```
./array1.sh  
unix  
unix linux program minix
```

2. Initializing An Array During Declaration

- Instead of initializing an each element of an array separately ,you can declare and initialize an array by specifying the list of elements (separated by white space)within a curly braces.

Syntax: declare –a arrayname=(element1 element2...)

- If the elements has the white space character, enclose it with in a quotes.

Example 2

```
#!/bin/bash
```

```
$cat array2.sh
```

```
declare -a os=('unix' 'linux' 'minix' 'aix');
```

- declare -a declares an array and all the elements in the parentheses are the elements of an array.

3.Print The Whole Bash Array

- There are different ways to print the whole elements of the array.
- If the index number is @ or *, all members of an array are referenced.
- You can traverse through the array elements and print it, using looping statements in bash.
echo \${os[@]}

Add the above echo statement into the array1.sh

#./t.sh

unix linux program minix

4. Length Of The Bash Array

- The length of an array can be displayed using the special parameter called \$#.
- **`${#arrayname[@]}`** gives you the length of the array.

```
$ cat array3.sh
```

```
declare -a Os=('Unix' 'linux' 'minix' 'aix');
```

```
echo ${#Os[@]} #Number of elements in the array
```

```
echo ${#Os}      #Number of characters in the first  
                  element of the array.i.e Unix
```

```
$/array3.sh
```

```
4
```

```
6
```


5. Length Of The nth Element In An Array

- `${#arrayname[n]}` should give the length of the nth element in an array.

```
$cat array4.sh
#!/bin/bash
emp[0]='Ravi'
emp[1]='Raj'
emp[2]='kishore'
emp[3]='aarthi'
echo ${#emp[3]} # length of the element located at index 3 i.e aarthi
```

```
$/array4.sh
```

6

6. Extraction by offset and length for an array

- The following example shows the way to extract 2 elements starting from the position 3 from an array called Unix.

```
$cat array5.sh
```

```
Os=('Unix' 'linux' 'minix' 'aix');
```

```
echo ${Os[@]:2:2}
```

```
$/array5.sh
```

```
minix aix
```

- The above example returns the elements in the 2nd index and 3rd index. Index always starts with zero.

7. Extraction with offset and length, for a particular element of an array

- To extract only first four elements from an array element.
- For example, minix which is located at the second index of an array, you can use offset and length for a particular element of an array.

```
$cat array6.sh
```

```
#!/bin/bash
```

```
Os= ('Unix' 'linux' 'minix' 'aix');
```

```
echo ${Os[2]:0:4}
```

```
./array6.sh
```

```
mini      # extracts the first four characters from the 2nd  
           indexed element
```

8. Search and Replace in an array elements

- The following example, searches for Suse in an array elements, and replace the same with the word 'SCO Unix'.

```
$cat array7.sh
```

```
#!/bin/bash
```

```
Place=('Chennai' 'Pune' 'Bangalore' 'kolkatta' 'Mumbai');
```

```
echo ${Place[@]/kolkatta/Delhi}
```

```
$./array7.sh
```

```
Chennai Pune Bangalore Delhi Mumbai
```

- But this example will not permanently replace the array content.

9. Add an element to an existing Bash Array

- The following example shows the way to add an element to the existing array.

```
$cat array8.sh
```

```
Place=('Chennai' 'Pune' 'Bangalore' 'kolkatta' 'Mumbai');
```

```
Place=("${Place[@]}" "Delhi")
```

```
echo ${Place[5]}
```

```
$/array8.sh
```

```
Delhi
```

- In the array called Place, the elements 'Delhi' is added in 5th index.

10. Remove an Element from an Array

- unset is used to remove an element from an array.
- unset will have the same effect as assigning null to an element.

```
$cat array9.sh
```

```
#!/bin/bash
```

```
Place=('Chennai' 'Pune' 'Bangalore' 'kolkatta' 'Mumbai');
```

```
unset Place[3]
```

```
echo ${Place[3]}
```

- The above script will just print null which is the value available in the 3rd index.

Cont...

- The following example shows one of the way to remove an element completely from an array.

```
$ cat array10.sh
```

```
Place=('Chennai' 'Pune' 'Bangalore' 'kolkatta' 'Mumbai'  
      'cochin' 'Madurai');
```

```
pos=3
```

```
Place=(${Place[@]:0:$pos} ${Place[@]:$(( $pos + 1))})
```

```
echo ${Place[@]}
```

```
$./array10.sh
```

```
Chennai Pune Bangalore Mumbai cochin Madurai
```

Cont...

- In this example, `${Place[@]:0:$pos}` will give you 3 elements starting from 0th index i.e 0,1,2
- `${Place[@]:4}` will give the elements from 4th index to the last index.
- And merge both the above output. This is one of the workaround to remove an element from an array.

11. Remove Bash Array Elements using Patterns

- In the search condition you can give the patterns, and stores the remaining element to an another array as shown below.

```
$ cat array11.sh
```

```
#!/bin/bash
```

```
declare -a Os=('Unix' 'linux kernel' 'minix' 'aix');
```

```
declare -a pattern=( ${Os[@]}/linux*/} )
```

```
echo ${pattern[@]}
```

```
$ ./array11.sh
```

```
Unix minix aix
```

```
#removes the pattern linux*
```

12. Copying an Array

- Expand the array elements and store that into a new array as shown below.

```
#!/bin/bash
```

```
Os=('Linux' 'Unix' 'Minix' 'aix');
```

```
kernel=("${Os[@]}")
```

```
echo ${kernel[@]}
```

```
$ ./array12.sh
```

```
Linux Unix  Minix  aix
```

13. Concatenation of two Bash Arrays

- Expand the elements of the two arrays and assign it to the new array.

```
$cat array13.sh
```

```
#!/bin/bash
```

```
Os=('Linux' 'Unix' 'Minix' 'aix');
```

```
Place=('Chennai' 'Pune' 'Bangalore' 'kolkatta' 'Mumbai');
```

```
Place2=("${Os[@]}" "${Place[@]}")
```

```
echo ${Place2[@]}
```

```
echo ${#Place2[@]}
```

```
$ ./array13.sh
```

```
Linux Unix Minix aix Chennai Pune Bangalore kolkatta
```

```
Mumbai #prints both the array elements
```

```
9 #no of elements in the new array
```

14. Deleting an Entire Array

- unset is used to delete an entire array.

```
$cat array14.sh
```

```
#!/bin/bash
```

```
Os=('Linux' 'Unix' 'Minix' 'aix');
```

```
Place=('Chennai' 'Pune' 'Bangalore' 'kolkatta' 'Mumbai');
```

```
Place2=("${Os[@]}" "${Place[@]}")
```

```
unset Place
```

```
echo "${#Place[@]}"
```

```
$ ./array14.sh
```

```
0      #After unset an array, its length would be zero as shown  
       above.
```

15. Load Content of a File into an Array

- You can load the content of the file line by line into an array.

#Example file

\$ cat logfile

Welcome to

programming

Linux

Unix

Cont...

```
$ cat loadcontent.sh
```

```
#!/bin/bash
```

```
filecontent=( `cat "logfile" `)
```

```
for t in "${filecontent[@]}"
```

```
do
```

```
echo $t
```

```
done
```

```
echo "Read file content!"
```

```
$ ./loadcontent.sh
```

Welcome to

programming

Linux

Unix

Read file content!

Example1

Declaring an array and assigning values

Syntax:-

#-----

#Arrayname[index]=value

os[0]=unix

os[1]=10

os[3]="/etc/passwd"

echo \${os[1]} # To access an element from an array

echo \${os[3]} #\${arrayname[index]}

./arr.sh

10

/etc/passwd

Example 2

```
os[0]="Linux kernel"
os[1]="Qnx Micro kernel"
SH[0]="/bin/sh"
SH[1]="/bin/bash"
echo ${os[0]}
echo ${SH[0]}
echo ${os[@]}
echo "Total:${#os[@]}"
c='expr ${#os[@]} - 1'
echo ${os[$c]}
```

```
# Print list of all the os names
# like $@   echo ${os[*]}   Like $*
# in command line args:$#
```

output will be as follows:

```
linux kernel
/bin/sh
Linux kernel Qnx Micro kernel
Total:2
Qnx Micro kernel
```


Example 3

```
Depts=(sales HR CRM )  
echo ${Depts[@]}  
echo # empty line  
for var in ${Depts[@]}  
do  
    echo "$var"  
done
```

Output will be:

```
./arrppt.sh  
sales HR CRM  
sales  
HR  
CRM
```

Example 4

```
read -p "Enter your file name :" fname
```

```
Array=(`ls -l $fname`)
```

```
for var in ${Array[@]}
```

```
do
```

```
    echo $var
```

```
done
```

```
echo -e "File name:${Array[7]}\tSize:${Array[4]}bytes"
```

Output:./arrppt.sh

Enter your file name :array3.sh

-rwxr-xr-x

1

root

root

77

2015-08-21

21:09

array3.sh

File name:array3.sh Size:77bytes

Example 5

```
IP=("127.0.0.1" "192.168.237.128")
```

```
for i in ${IP[@]}
```

```
do
```

```
    ping -c 2 $i
```

```
done >>$1          # output redirected to a runtime argument file
```

Output: ./arrppt.sh kk.sh

```
cat kk.sh
```

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
```

```
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.063 ms
```

```
64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.051 ms
```

```
--- 127.0.0.1 ping statistics ---
```

```
2 packets transmitted, 2 received, 0% packet loss, time 999ms
```

```
rtt min/avg/max/mdev = 0.051/0.057/0.063/0.006 ms
```

```
PING 192.168.237.128 (192.168.237.128) 56(84) bytes of data.
```

```
From 192.168.1.100 icmp_seq=2 Destination Host Unreachable
```

```
--- 192.168.237.128 ping statistics ---
```

```
2 packets transmitted, 0 received, +1 errors, 100% packet loss, time 1008m
```

Example 6

```
IP=("127.0.0.1" "192.168.237.128")
```

```
for i in ${IP[@]}
```

```
do
```

```
    read -p "Your checking $i IPaddress Enter count value:" n
```

```
    ping -c $n $i
```

```
    echo
```

```
    "
```

```
    "
```

```
done >>$1
```

- In the above program, the IP address have been sent through for loop and the result is redirected to the runtime argument file.

Output :

./Ip.sh k2.sh

Your checking 127.0.0.1 IPaddress Enter count value:2

Your checking 192.168.237.128 IPaddress Enter count value:2

root@ubuntu:~/Sangeetha# cat k2.sh

PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.

64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.057 ms

64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.045 ms

--- 127.0.0.1 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 999ms

rtt min/avg/max/mdev = 0.045/0.051/0.057/0.006 ms

PING 192.168.237.128 (192.168.237.128) 56(84) bytes of data.

From 192.168.1.100 icmp_seq=2 Destination Host Unreachable

--- 192.168.237.128 ping statistics ---

2 packets transmitted, 0 received, +1 errors, 100% packet loss, time 1008ms

Example 7

```
# Copying an array
employee=("sara" "kevin" "mohamad")
echo "(A)">${employee[@]}"
echo "(B)">${#employee[@]}"
ID=("${employee[@]}")          # copying from employee array to ID
echo "(C)">${ID[@]}"
echo "(D)">${#ID[@]}"
echo -e "(E)">${ID[0]}\t${ID[2]}"
```

Output ./cpy.sh

Sara kevin mohamad

3

Sara kevin mohamad

3

Sara mohamad

Example 8

```
# Concatenation of two arrays
lang=(Java c c++)
OS=(unix linux winx minix aix)
echo -e "(A)..${SH[@]}\t Size:${#SH[@]}\n" echo -e
"(B)..${OS[@]}\t Size:${#OS[@]}\n"
Array1=("${SH[@]}" "${OS[@]}") # Concatentation
echo -e "(C)..${Array1[@]}\t Size:${#Array1[@]}\n"
```

Output: ./concate.sh

(A)..Java c c++ Size:3

(B)..unix linux winx minix aix Size:5

(C)..Java c c++ unix linux winx minix aix Size:8

Example 9

```
# Add an element to an existing array
os=(unix linux aix minix)
echo -e "(A)...${os[@]}\t Size:${#os[@]}"
#os=("Winxp" "Hpux" "${os[@]}" "Winx7")
os=("${os[@]}" "Solaris" "Oracle Linux")
echo -e "(B)...${os[@]}\t Size:${#os[@]}"
echo "(C). ${os[5]}"
```

Output : ./add.sh

(A)...unix linux aix minix Size:4

(B)...unix linux aix minix Solaris Oracle Linux Size:6

(C). Oracle Linux

Example

```
1 0 1 0
os=(unix linux minix aix winx)
echo -e "(A)..List of all:${os[@]}\t Size:${#os[@]}\n"
echo -e "(B)..1st index:${os[1]}\t 2nd index:${os[2]}\n"
OS1=("${os[@]}")
unset os[1]    # To delete an array element, use unset command
unset os[2]
echo "After unset:"
echo -e "(C)..List of all:${os[@]}\t Size:${#os[@]}\n"
echo -e "(D)..1st index:${os[1]}\t 2nd index:${os[2]}\n"
echo -e "\n..OS1: ${OS1[@]}\t Size:${#OS1[@]}"
```

Output: ./arrppt.sh

```
(A)..List of all:unix linux minix aix winx      Size:5
(B)..1st index:linux      2nd index:minix
After unset:
(C)..List of all:unix aix winx      Size:3
(D)..1st index:      2nd index:
..OS1: unix linux minix aix winx      Size:5
```

Example 11

```
os='uname -sr'
echo "(1)::$os"
unset os
os='uname -sp'
echo "(2)::$os"
SH=$SHELL
echo "(3)::$SH"
echo "Working Shell name:$SH"
echo "$SH version is
    $BASH_VERSION"
unset SH
echo "After unset..."
echo "(3)::$SH"
echo "Working Shell name:$SH"
echo "$SH version is
    $BASH_VERSION"
```

#output will be:

```
./arrppt.sh
(1)::.Linux 3.0.0-12-generic
(2)::.Linux i686
(3)../bin/bash
Working Shell name:/bin/bash
/bin/bash version is 4.2.10(1)-release
After unset...
(3)..
Working Shell name:
    version is 4.2.10(1)-release
```

Example

- Extraction with offset and length, for a particular element of an array

Example:

```
os=(unix linux winx minix aix "oracle linux" Hpux)
```

```
echo ${os[@]:1:4}
```

```
# echo ${os[@]:0:2}
```

```
# offset --> index
```

```
# length
```

```
# echo ${os[2]:1:3}
```

Example 13

Search and Replace in an array elements

```
os=("unix-os" "linux-shell" "unix-essentials" "open unix concepts" "system  
programs in unix" "Winx os")
```

```
echo "(A)..\${os[@]}"
```

```
echo "(B)..\${os[0]/unix/KERNEL}"
```

Search Replace

```
Newos=("${os[@]/unix/KERNEL}")
```

```
echo "(C)..\${os[@]}"
```

```
echo "(D)..\${Newos[@]}"
```

```
for i in ${Newos[@]}
```

```
do
```

```
    echo $i
```

```
done
```

Example 15

Search and Replace in an array elements

```
os=("unix-os" "linux-shell" "unix-essentials" "open unix concepts" "system  
programs in unix" "Winx os")
```

```
echo "(A)..\${os[@]}"
```

```
echo "(B)..\${os[0]/unix/KERNEL}"
```

#Search Replace

```
echo "${os[@]:0:3}" # /unix/KERNEL"
```

```
A=("${os[@]:0:3}")
```

```
echo "${A[@]/unix/KERNEL}";
```

Example 14

```
Array=($@)                #runtime argument
for var in ${Array[@]}
do
    echo $var
done
echo "Count:${#Array[@]}"
echo
echo "Count:$#"

```

Output:

```
./arrppt.sh 10 20 30 40
10
20
30
40
Count:4

```

Example 16

```
echo "Enter Emp details:"
read -a emp
echo "Input details.."
count=0
for i in ${emp[@]}
do
    echo "Index:$count Value is ${emp[$count]}"
    count=`expr $count + 1`
done
```

Output will be:

```
./arrppt.sh
Enter Emp details:
sara 10 chennai
Input details..
Index:0 Value is sara
Index:1 Value is 10
Index:2 Value is chennai
```


Example

[illegible]

Example 18

[illegible]

Example 19

```
readonly -a shells=("ksh" "bash" "sh" "csh" "tcsh")
```

```
#shells=("ksh" "bash" "sh" "csh" "tcsh")
```

```
echo "${shells[@]}"
```

```
echo "${#shells[@]}"
```

```
shells[0]="Gnu-Bash" #as it is a read only variable,no change will occur.
```

```
echo "${shells[@]}" # same as previous echo statement
```

```
echo "${#shells[@]}"
```

Example 20

```
declare -i var;  #will accept only integer values.
```

```
var=10
```

```
echo $var
```

```
var=200
```

```
echo $var
```

```
var=1.344      # will show error message
```

```
echo $var
```

Functions

What is function ?

- Functions are nothing but collection of statements or collection of program.
- when ever we call the function ,it will perform set of task.
- In shell script function is defined as collection of commands or subroutines or script with in sub scripts.
- Function will improves script readability.
- This will improves to break up a complex script into separate tasks, so it's easy to debug.

- Shell functions act as "**scripts with in another scripts**" and allow us to follow top-down design principles.
- We are not following function prototype (unlike C language) so function definition should appear 1st section (function section)
- Function call should start from Script section.

Function Declaration

- To declare shell function we should follow below syntax:

```
function FunctionName()
```

```
{
```

```
    Function Definition
```

```
}
```

- The function body - the portions of the function between the { and } characters.
- Function name should enclose ()
- **function** is a key word.

1. **function** function_name()
2. {
3. function operations
4. } # exit from function block
5. echo "Script will start from this line"
6. function_name # function call
7. echo "Exit from \$0"

- From the above script, shell 1st will interpret 5th line then 6th line (function call) then control will go to line 1 (function block will start)
- it will exit at line number 4.
- After complete 4th line script block will be enabled, script exit with 7th line.
- **Note: \$0 - Display Script Name**

Example :1

```
function Hello()  
{  
    echo "This is Hello function block.."  
}
```

Hello # Function call

```
echo "Exit from $0"
```

Example:2

```
Hello(){
```

```
  echo "This is Hello function block..“
```

```
  }
```

```
sales(){
```

```
  echo "Sales block..“
```

```
}
```

```
Hello # Function call
```

```
sales # Function call
```

```
echo "Exit from $0"
```

Example:3

```
Hello(){  
  echo "This is Hello function block.."  
  echo "Exit from $FUNCNAME"  
}  
  
sales(){  
  set -x  
  
  echo "Sales block.."  
  
  echo "Exit from $FUNCNAME"  
  
  set +x  
}  
  
# Function call sales  
  
# Function call  
  
echo "Exit from $0"
```

Example:4

```
Display(){
echo "List of files:" $(ls )
echo #empty line
sysinfo # nested Function
echo #empty line
diskusage # nested Function
echo "Exit From $FUNCNAME"
}
sysinfo()
{
echo "System Information:"
echo "kernel name:$(uname)"
echo "Shell name:$SHELL"
echo "Exit From $FUNCNAME"
}
diskusage()
{
echo "Disk Usage:" $(du)"
echo "Exit From $FUNCNAME"
}
echo Display # Function call
echo "Exit from $0"
```

Example:5

```
Hello(){
```

```
    echo "This is $FUNCNAME"
```

```
    echo $1 $2 $3
```

```
    echo $@
```

```
    echo "Total:$#"
```

```
}
```

```
Hello 10 20 3.5 T unix # function with args call
```

Example:6

Hello()

```
{  
  echo "This is $FUNCNAME"  
  echo $1 $2 $3  
  echo $@  
  echo "Total:$#"  
  echo "Exit from $FUNCNAME"  
}
```

Dispaly()

```
{  echo "This is $FUNCNAME"; echo "$1 $2"  
  echo $@ ; echo $#  
  echo "Exit from $FUNCNAME"  
}
```

echo -e "Script section \n"Hello 10 20 3.5 T unix # function with args call

Display Ram Sales Bangalore 4567.89

echo -e "\n Exit from \$0"

Example:7

```
Hello(){  
    v1=10 v2=1.5    v3=Unix  
  
    Print $v1 $v2 $v3 #nested function  
  
    echo "Exit from $FUNCNAME"  
  
}  
  
Print(){  
  
    echo $1 $2 $3  
  
    echo "$@"  
  
}  
  
Hello
```

Example:8

```
Getdata(){
    echo -e "Name:$1 \t Dept:$2"
    Print $1 $2 "Bangalore"
    echo "Total:$#"
    echo "Exit From $FUNCNAME"
}

Print(){
    echo -e "Name:$1 \t Dept:$2 \t Place:$3"
    echo "Total:$#"
    echo "Exit From $FUNCNAME"
}

echo "Enter emp name and dept"
read name;read dept
Getdata $name $dept
echo "Exit From $0"
```

Example:9

Print()

{

echo -e "Name:\$1 \t Dept:\$2 \t Place:\$3"

echo \$#

}

Print \$@

bash p9.sh **Ram sales Pune**

Example:10

```
Print(){  
    echo -e "Name:$1 \t Dept:$2 \t Place:$3"  
    echo $#  
}
```

```
Getdata(){  
    echo "Student Details:"  
    echo -e "Name:$1 \t Dept:$2 \t Place:$3"  
}
```

Print \$@

```
echo "Enter Student Name and Dept"
```

```
read name;read dept
```

Getdata \$name \$dept \$3

Example:11

```
File_Test(){
```

```
if [ -e $1 ];then
```

```
    echo "$1 is available"
```

```
else
```

```
    echo "Sorry $1 file is not available"
```

```
fi
```

```
}
```

File_Test \$1 # Function with argument,it will check file is existsing or not.

Example:12

```
sum(){  
  sum='expr $1 + $2'  
  echo $sum  
}  
sum 10 20
```

Example:13

```
Sum(){  
  sum='expr $1 + $2'  
  return $sum  
}  
  
Sum 10 20  
  
echo "Sum:$sum"
```

local keyword

- local variables
- A variable declared as local is one that is visible only within the block of code in which it appears.
- It has local scope.
- In a function, a local variable has meaning only within that function block

Example:14

```
sum()
{
local sum='expr $1 + $2' # variable sum is visible only within the block of code
return $sum
}

sum 10 20

echo "Sum:$sum" # print empty line ,

# because sum is a local variable not visible to out of code block.
```

Example:15

```
One()
{
local r1='expr 10 + 20'
return $r1
}
Two(){
local r2='expr 10 \* 20'
return $r2
}
```

One

Two

echo \$? ; echo \$?

Example:16

```
One(){  
    local r1='expr 10 + 20'  
    return $r1  
}
```

```
Two(){  
    local r2='expr 10 \* 20'  
    return $r2  
} One
```

```
echo $?
```

```
Two
```

```
echo $?
```

Example:17

```
Hello(){  
    echo "$1 $2 $3"  
    echo "Total:$#"   
    echo "List of all :$@"  
    echo "Exit from $FUNCNAME"  
}  
  
sales(){  
    echo "$1 $2"  
    echo $#  
    echo $@  
    echo "Exit from $FUNCNAME"  
}  
  
echo "Script section.."  
  
Hello $1 $2 # function with args.  
  
sleep 3  
  
echo sales $@  
  
echo "Exit from $0"
```

Example:18

```
Getdata(){  
    read -p "Enter Emp.ID:" id  
    read -p "Enter Name:" name  
    read -p "Enter working place:" place  
    Display $id $name $place "sales"  
    echo "Exit from $FUNCNAME"  
    }Display(){  
    echo -e "ID:$1 \t Name:$2 \t Place:$3"  
    echo "Mr.$2 working dept is $4"  
    echo "Exit from $FUNCNAME"  
    }  
    echo # Empty line  
    Getdata  
    echo -e "\nExit from $0"
```

Example:19

```
calc(){  
  v1=$1 v2=$2  
  local sum='expr $v1 + $v2'  
  # sum='expr $1 + $2'  
  #echo "Total:$sum"  
  return $sum  
}  
  
calc $@  
  
echo "total:$?"  
  
#echo "Total:$sum"  
  
echo "Exit from $0"
```

Example

Demonstration of a simple recursive function

- RECURSIONS=9 # How many times to recurse.
- r_count=0 # Must be global. Why?
- recurse ()
- {
- var="\$1"
- while ["\$var" -ge 0]
- do
- echo "Recursion count = "\$r_count" +--+ \"\$var = \"\$var""
- ((var--)); ((r_count++))
- recurse "\$var" # Function calls itself (recurses)
- done # until what condition is met?
- }
- **recurse** \$RECURSIONS
- exit \$?

THANK YOU