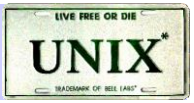
# Function and Array

## Functions

- Functions make scripts easier to maintain. Basically it breaks up the program into smaller pieces. A function performs an action defined by you, and it can return a value if you wish.

```
#!/bin/bash
  hello()
  {
  echo “You are in function hello()”
  }

  echo “Calling function hello()…”
  hello
  echo “You are now out of function hello()”
```

- In the above, we called the hello() function by name by using the line:  hello . When this line is executed, bash searches the script for the line hello(). It finds it right at the top, and executes its contents.




## Functions

- A shell function is similar to a shell script
  - stores a series of commands for execution later
  - shell stores functions in memory
  - shell executes a shell function in the same shell that called it

- must be defined before they can be referenced
- usually placed at the beginning of the script

Syntax:

```
function-name () {
    statements
}
```

## Example: function

```
#!/bin/bash

funky () {
  # This is a simple function
  echo "This is a funky function."
  echo "Now exiting funky function."
}

# declaration must precede call:

funky
```

## Functions

```
$ cat function.sh
#!/bin/bash
function check() {
if [ -e "/home/$1" ]
then
  return 0
else
  return 1
fi
}
echo “Enter the name of the file: ” ; read x
if check $x
then
  echo “$x exists !”
else
  echo “$x does not exists !”
fi.
```

## Example: Picking a random card from a deck

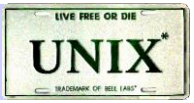
```
#!/bin/bash
# Count how many elements.

Suites=“Clubs Diamonds Hearts Spades”
Denominations=“2 3 4 5 6 7 8 9 10 Jack Queen King Ace”

# Read into array variable.
suite=($Suites)
denomination=($Denominations)

# Count how many elements.
num_suites=${#suite[*]}
num_denominations=${#denomination[*]}
echo -n "${denomination[$((RANDOM%num_denominations))]} of "
echo ${suite[$((RANDOM%num_suites))]}
exit 0
```

## Example: Changes all filenames to lowercase

```
#!/bin/bash
for filename in *
                                # Traverse all files in directory.
do
                                # Get the file name without the path.
 fname=`basename $filename`
                                # Change name to lowercase.
 n=`echo $fname | tr A-Z a-z`
 if [ “$fname” != “$n” ]
                                # Rename only files not already lowercase.
 then
   mv $fname $n
 fi
done
exit 0
```

## Example: Compare two files with a script

```
#!/bin/bash
ARGS=2                                  # Two args to script expected.
if [ $# -ne “$ARGS” ]; then
  echo “Usage: `basename $0` file1 file2” ; exit 1
fi
if [[ ! -r "$1" || ! -r "$2" ]] ;  then
  echo “Both files must exist and be readable.” ; exit 2
fi
                                        # /dev/null buries the output of the “cmp” command.
cmp $1 $2 &> /dev/null
                                        # Also works with 'diff', i.e., diff $1 $2 &> /dev/null
if [ $? -eq 0 ]                         # Test exit status of “cmp” command.
then
  echo “File \“$1\” is identical to file \“$2\”.”
else
  echo “File \“$1\“ differs from file \“$2\”.”
fi
exit 0
```




## Function parameters

- Need not be declared
- Arguments provided via function call are accessible inside function as $1, $2, $3, …

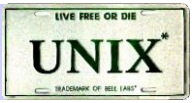
| | |
|---|---|
| $# | reflects number of parameters |
| $0 | still contains name of script (not name of function) |

## Example: function with parameter

```
#! /bin/sh
testfile() {
  if [ $# -gt 0 ]; then
     if [[ -f $1 && -r $1 ]]; then
        echo $1 is a readable file
     else
        echo $1 is not a readable file
     fi
  fi
}

testfile .
testfile funtest
```

## Example: function with parameters

```
#! /bin/bash
checkfile() {
    for file
    do
        if [ -f "$file" ]; then
            echo "$file is a file"
        else
            if [ -d "$file" ]; then
                echo "$file is a directory"
            fi
        fi
    done
}
checkfile . funtest
```




## Local Variables in Functions

- Variables defined within functions are global,
  i.e. their values are known throughout the entire shell program

- keyword “local” inside a function definition makes referenced
  variables “local” to that function

## Example: function

```
#! /bin/bash

global="pretty good variable"

foo () {
        local inside="not so good variable"
        echo $global
        echo $inside
        global="better variable"
}

echo $global
foo
echo $global
echo $inside
```

# Using Arrays with Loops

- In the bash shell, we may use arrays. The simplest way to create one is using one of the two subscripts:

```
pet[0]=dog
pet[1]=cat
pet[2]=fish
pet=(dog cat fish)
```

- We may have up to 1024 elements. To extract a value, type ${arrayname[i]}

```
$ echo ${pet[0]}
  dog
```

- To extract all the elements, use an asterisk as:

```
echo ${arrayname[*]}
```

- We can combine arrays with loops using a for loop:

```
for x in ${arrayname[*]}
  do
            ...
  done
```

## Debugging

- Bash provides two options which will give useful information for debugging

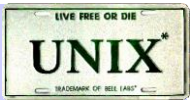
-x : displays each line of the script with variable substitution and before execution
-v : displays each line of the script as typed before execution

- Usage:

#!/bin/bash –v or #!/bin/bash –x or #!/bin/bash –xv

$ cat for3.sh

```
#!/bin/bash –x
echo –n “Enter a number: ”; read x
let sum=0
for (( i=1 ; $i<$x ; i=$i+1 )) ; do
  let “sum = $sum + $i”
done
echo “the sum of the first $x numbers is: $sum”
```




## Debugging

```
$ for3.sh
+ echo –n ‘Enter a number: ’
Enter a number: + read x
3
+ let sum=0
+ (( i=0 ))
+ (( 0<=3 ))
+ let ‘sum = 0 + 0’
+ (( i=0+1 ))
+ (( 1<=3 ))
+ let ‘sum = 0 + 1’
+ (( i=1+1 ))
+ (( 2<=3 ))
+ let ‘sum = 1 + 2’
+ (( i=2+1 ))
+ (( 3<=3 ))
+ let ‘sum = 3 + 3’
+ (( i=3+1 ))
+ (( 4<=3 ))
+ echo ‘the sum of the first 3 numbers is: 6’
the sum of the first 3 numbers is: 6
```

# Example: Suite drawing statistics

```
$ cat cardstats.sh
#!/bin/sh # -xv
N=100000
hits=(0 0 0 0)# initialize hit counters
if [ $# -gt 0 ]; then           # check whether there is an argument
        N=$1
else                            # ask for the number if no argument
        echo "Enter the number of trials: "
        TMOUT=5                 # 5 seconds to give the input
        read N
fi
i=$N
echo "Generating $N random numbers... please wait."
SECONDS=0                       # here is where we really start
while [ $i -gt 0 ]; do  # run until the counter gets to zero
        case $((RANDOM%4)) in                           # randmize from 0 to 3
                0) let "hits[0]+=1";;                   # count the hits
                1) let "hits[1]=${hits[1]}+1";;
                2) let hits[2]=$((${hits[2]}+1));;
                3) let hits[3]=$((${hits[3]}+1));;
        esac
        let "i-=1”# count down
done
echo "Probabilities of drawing a specific color:"
                                # use bc - bash does not support fractions
echo "Clubs: " `echo ${hits[0]}*100/$N | bc -l`
echo "Diamonds: " `echo ${hits[1]}*100/$N | bc -l`
echo "Hearts: " `echo ${hits[2]}*100/$N | bc -l`
echo "Spades: " `echo ${hits[3]}*100/$N | bc -l`
echo "========================================="
echo "Execution time: $SECONDS"
```