Torry Harris

Arrays example

#Using arrays

```
city=("mumbai" "kolkata" "delhi“ "chennai“)
echo “Value at first index is ${city[0]}“
city[0]="Bangalore"
echo “Changed value at first index is ${city[0]}"
echo "Array size is ${#city[*]}“
city[4]="Chandigarh"
```

Torry Harris

Arrays Example

```
echo "Array size is ${#city[*]}"
    s=${#city[*]}
    i=0
    while [ $i -lt $s ]
    do
        echo "Value at index $i is  ${city[i]}"
        i=`expr $i + 1`
    done
echo "All the array items are ${city[*]}
list=($1 $2 $3)
echo "All the array items are ${list[*]}
```

Positional Parameters

- Parameters can be passed ***from the command line*** to a shell script
- A parameter is any word or a string which is specified along with a command name or file-name
- You can pass parameters to scripts which can be referred to by ***position*** within the script.
- The shell substitutes the values of the positional parameters before running the command
- UNIX offers a set of ***9*** positional parameters starting from ***$1 to $9***
- The name of the script is held in the ***$0 positional parameter.***
- ***$#*** stores the **count** of the positional parameters
- ***$**** stores the **string** of positional parameter values

Torry Harris

Positional Parameters

$1,$2	Positional parameter
$0	Command name
$#	Count of args
$*	Complete set of positional parameters as a string
“$@” (remember the quotes)	Here, each quoted string is treated as a separate argument

Torry Harris

Example

debalina@bootcamp:~

```
echo -e "\n Script name is $0"
echo -e "\n Count of arguments is $#"
echo -e "\nString of args is $*"

echo -e "\n First file is $1"
echo -e "\n Second file is $2"

#Concatenating files  into a third file

cat $1 $2 > newfile

#Displaying data from the 3rd file

cat newfile
~
```

Torry Harris

Example
(using $@)

```
#Using  “$@”
n=0
for i  in  "$@”
do
 n=`expr $n + 1`
echo $n
echo $i
done
```

Torry Harris

Example
(Using $*)

```
#Using  “$*”
n=0
for i in  "$*”
do
n=`expr $n + 1`
echo $n
echo $i
done
```

Torry Harris

Shifting Positional Parameters

- We can only refer directly to 9 positional parameters from $1 to $9($0 contains the file name)
- However, there may be situations wherein there are more than 9 arguments ,or the user is allowed to key in variable number of arguments
- We can "pull" parameters after the 9th into view with the ***shift command***

After a shift

$1 moves out

$2 -> $1

....

${10} -> $9

When the shift command is executed ,the value of $1 at that point in time is lost, and the value of $2 is put into $1 .In the same way the value of $3 is moved to $2, and so on.

The value of $# will be reduced by 1,and the first word in the string contained in $* will be deleted after every shift

Torry Harris

Example
(using shift)

debalina@bootcamp:~

```
#Using the "shift" command to calculate the sum of variable number of arguments

echo -e "\n Count of numbers is $#"

 while test $# -gt 0
 do
   echo -e "\n \$1 value is $1"
   sum=`expr $sum + $1`
   shift
 done
echo -e "\n Sum is $sum"
```

debalina@bootcamp:~

```
[debalina@bootcamp ~]$ sh p2.sh 3 8 10

 Count of numbers is 3

 $1 value is 3

 $1 value is 8

 $1 value is 10

 Sum is 21
[debalina@bootcamp ~]$
```

Sample output with 3 numbers

Torry Harris

Use of
((and))
operators

- The operators ((and)) is an alternative way to perform arithmetic computation
- Is likely to become a standard feature of the shells
- Is easy to use
- Supports all arithmetic operations
- Its easy as each variable need not have to be preceded by a $ symbol
- Please note the entire arithmetic operation however needs to be preceded by a single $ symbol.

Torry Harris

Example

```
#Script using (( and )) operator
a=200 b=2
c=12
c=$((a+b+c))
echo $c
c=$((a/b))
echo $c
c=$((c%b))
echo $c
c=$((c+1))
echo $c
d=$((a\*b))
echo $d
```

Functions

- Functions enable you to break down the overall functionality of a script into smaller, logical subsections.
- These subsections can then be called upon to perform their individual task when it is needed.
- Using functions to perform repetitive tasks is an excellent way to create code reuse.
- 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
        }
```

Torry Harris

Example –on function

```
#Defining a  function to count the contents of a directory
dir_count_disp(){
echo "enter directory name"
 read dname
 echo "Directory is $dname"
count=`ls –l $dname|wc -l`
 echo "Count of files under $dname is $count"
}
dir_count_disp
```

Torry Harris

Example –on Nested function

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

Torry Harris

Example

```
#Defining a  function with parameters
fn(){
echo "Function accessing positional parameters and
     other variables in script"
echo "fn \$1=$1"
echo "fn \$2=$2"
echo "List of paramters is $*"
echo "Count of paramters is $#"
echo "Accessing other variables in script"
echo "v=$v"
}
v=10
echo $v
fn 200 300
```


STOP