SHELL PROGRAMMING PART 3

Review of Basic UNIX familiar commands often useful in shell scripts

`cat`	concatenate files
`cp`	copy a file
`date`	print the date and time
`grep`	scan for a string
`head`	show first lines of a file
`tail`	show last lines of a file
`mv`	move or rename a file
`rm -f`	remove files (silently)
`wc`	count lines, words, characters
	`wc` output format varies between systems

Some New Commands Useful in Shell Programs

`basename`	extract file name from path name
`cmp -s`	compare files (silently)
`cut`	extract selected parts of a line
`expr`	evaluate an expression
`mail`	send email (not in `cygwin`)
`sed -e`	stream editor
`sleep`	suspend execution for given time
`tr`	translate characters
`true`, `false`	provide truth values
`whoami`	print current username
`head -1`	read a line from the keyboard

Shell supports a different type of variable called an **array variable**. This can hold multiple values at the same time. Arrays provide a method of grouping a set of variables. Instead of creating a new name for each variable that is required, you can use a single array variable that stores all the other variables.

All the naming rules discussed for Shell Variables would be applicable while naming arrays.

Defining Array Values

The difference between an array variable and a scalar variable can be explained as follows.

Suppose you are trying to represent the names of various students as a set of variables. Each of the individual variables is a scalar variable as follows −

```
NAME01="Zara"

NAME02="Qadir"

NAME03="Mahnaz"

NAME04="Ayan"

NAME05="Daisy"
```

We can use a single array to store all the above mentioned names. Following is the simplest method of creating an array variable. This helps assign a value to one of its indices.

```
array_name[index]=value
```

Here *array_name* is the name of the array, *index* is the index of the item in the array that you want to set, and value is the value you want to set for that item.

Accessing Array Values

After you have set any array variable, you access it as follows –

```
${array_name[index]}
```

Here *array_name* is the name of the array, and *index* is the index of the value to be accessed. Following is an example to understand the concept –

Live Demo

```
#!/bin/sh



NAME[0]="Zara"

NAME[1]="Qadir"

NAME[2]="Mahnaz"

NAME[3]="Ayan"

NAME[4]="Daisy"

echo "First Index: ${NAME[0]}"

echo "Second Index: ${NAME[1]}"
```

The above example will generate the following result –

```
$./test.sh
First Index: Zara
Second Index: Qadir
```

You can access all the items in an array in one of the following ways –

```
${array_name[*]}
${array_name[@]}
```

Here **array_name** is the name of the array you are interested in. Following example will help you understand the concept –

Live Demo

```
#!/bin/sh



NAME[0]="Zara"

NAME[1]="Qadir"

NAME[2]="Mahnaz"

NAME[3]="Ayan"

NAME[4]="Daisy"

echo "First Method: ${NAME[*]}"

echo "Second Method: ${NAME[@]}"
```

The above example will generate the following result –

```
$./test.sh
First Method: Zara Qadir Mahnaz Ayan Daisy
Second Method: Zara Qadir Mahnaz Ayan Daisy
```

Example:

```
#!/bin/bash
NAME[0]="Zara"
NAME[1]="Qadir"
NAME[2]="Mahnaz"
NAME[3]="Ayan"
NAME[4]="Daisy"
echo "First Index: ${NAME[0]}"
echo "Second Index: ${NAME[1]}"
echo "Third Index: ${NAME[2]}"
echo "4th Index: ${NAME[4]}"
```

We will now discuss the following operators –

- Arithmetic Operators
- Relational Operators
- Boolean Operators
- String Operators
- File Test Operators

Bourne shell didn't originally have any mechanism to perform simple arithmetic operations but it uses external programs, either **awk** or **expr**.

The following example shows how to add two numbers −

Live Demo

```
#!/bin/sh


val=`expr 2 + 2`

echo "Total value : $val"
```

The above script will generate the following result −

```
Total value : 4
```

The following points need to be considered while adding −

- There must be spaces between operators and expressions. For example, 2+2 is not correct; it should be written as 2 + 2.
- The complete expression should be enclosed between **‘ ‘**, called the backtick.

Arithmetic Operators

The following arithmetic operators are supported by Bourne Shell.

Assume variable **a** holds 10 and variable **b** holds 20 then −

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.

!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

It is very important to understand that all the conditional expressions should be inside square braces with spaces around them, for example **[$a == $b]**is correct whereas, **[$a==$b]** is incorrect.

All the arithmetical calculations are done using long integers.

Here is an example which uses all the arithmetic operators −

Live Demo

```
#!/bin/sh


a=10

b=20


val=`expr $a + $b`

echo "a + b : $val"


val=`expr $a - $b`

echo "a - b : $val"
```

```
val=`expr $a \* $b`

echo "a * b : $val"


val=`expr $b / $a`

echo "b / a : $val"


val=`expr $b % $a`

echo "b % a : $val"
```