# Basic Shell Programming

## 1. Variables

Creating variables in bash is similar to other languages. There are no data types. All variables are considered and stored as strings, even when they are assigned numeric values. You do not need to declare a variable, just assigning a value to its reference will create it.

Do not put spaces before or after the `=` sign.

Example:

```
var=3
```

The above line creates a variable `var` and assigns 3 to it. You can print a variable by putting the `$` sign before the name of the variable.

Example:

```
echo $var # 3
```

You can use the `read` command to take an input from user and store it into a variable

Example:

```
read a
echo $a
```

## 2. Array

Like other languages bash has also arrays. An array is a variable containing multiple values. There's no maximum limit on the size of array. Arrays in bash are zero based. The first element is indexed with element 0. There are several ways for creating arrays in bash which are given below.

Examples:

```
array[0]=val
array[1]=val
array[2]=val
array=([2]=val [0]=val [1]=val)
array=(val val val)
```

To display a value at specific index use following syntax:

```
${array[i]}     # where i is the index
```

If no index is supplied, array element 0 is assumed. To find out how many values there are in the array use the following syntax:

```
${#array[@]}
```

## 3. Expressions

```
expr 1 + 1 # 2
expr 1+1 # 1+1

var=expr 1 + 1    # command not found

var=`expr 1 + 1`
echo $var    # 2

var='expr 1 + 1'
echo $var    # expr 1 + 1

var="expr 1 + 1"
echo $var    # expr 1 + 1
```

Expression Examples:

```
statement1 && statement2  # both statements are true
statement1 || statement2  # at least one of the statements is true

str1=str2       # str1 matches str2
str1!=str2      # str1 does not match str2
str1<str2       # str1 is less than str2
str1>str2       # str1 is greater than str2
-n str1         # str1 is not null (has length greater than 0)
-z str1         # str1 is null (has length 0)

-lt     # less than
-le     # less than or equal
-eq     # equal
-ge     # greater than or equal
-gt     # greater than
-ne     # not equal
```

## 4. Conditionals

The conditional statement in bash is similar to other programming languages.

```
if [ expression ]; then
    will execute only if expression is true
else
    will execute if expression is false
fi
```

Example 1

```
read a
read b
if [ $a = $b ]; then
    echo a and b are  equal
```

```
else
    echo a and b are not equal
fi
```

Example 2

```
read a
if [ `expr $a % 2` = 0 ]; then
    echo a is even
else
    echo a is odd
fi
```

You can also use `case` statements for implementing conditionals.

```
case expression in
    pattern1 )
        statements ;;
    pattern2 )
        statements ;;
    ...
esac
```

```
read a
case `expr $a % 2` in
    0) echo even;;
    1) echo odd;;
    *) echo not valid input;;
esac
```

```
echo “Is it morning? Please answer yes or no”
read timeofday
case $timeofday in
    yes ) echo “Good Morning”;;
    no ) echo “Good Afternoon”;;
    y ) echo “Good Morning”;;
    n ) echo “Good Afternoon”;;
    * ) echo “Sorry, answer not recognized”;;
esac
```

## 5. Loops

There are three types of loops in bash. `for`, `while` and `until`.

`for` Syntax:

```
for name [in list]
do
  statements that can use $name
done
```

```
#!/bin/bash
for i in 1 2 3 4 5
do
   echo "$i"
done
```

```
for i in {1..5}; do
    echo $i
done
```

```
for i in {1..40..12}; do
    echo $i
done
```

`while` Syntax:

```
while condition; do
  statements
done
```

```
sum=0
while [ 1 ]
do
    read a
    if [ $a -eq 0 ];then
        break
    fi
    if [ $a -lt 0 ];then
        continue
    fi
    sum=`expr $sum + $a`
done
echo sum is $sum
```

## 6. Functions

As in almost any programming language, you can use functions to group pieces of code in a more logical way or practice the divine art of recursion. Declaring a function is just a matter of writing function my_func { my_code }. Calling a function is just like calling another program, you just write its name.

```
function name() {
    shell commands
}
```

Example:

```
#!/bin/bash
function hello {
   echo world!
}
hello
```

```
function say {
    echo $1
}
say "hello world!"
```

```
showarg(){
    a=1
    for i in $*
    do
        echo "The $a No arg is $i"
        a=`expr $a + 1`
    done
}
showarg 1 2 3
```