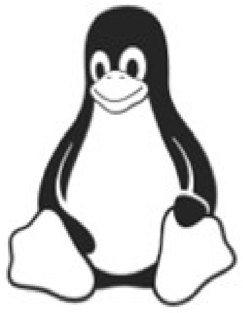
Linux™

Shell Scripting

What is Shell Scripting ?



A shell script is a computer program designed to be run by the Unix/Linux shell.

which could be one of the following:

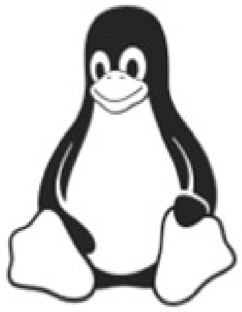
- The Bourne Shell
- The C Shell
- The Korn Shell
- The GNU Bourne-Again Shell

- A shell is a command-line interpreter and typical operations performed by shell scripts include

    file manipulation, program execution and printing text

- **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell Prompt



- The prompt $, which is called command prompt is issued by the shell. While the prompt is displayed, you can type a command.
- Example
- Shell command, which displays the current date and time is

```
$date
```

- #!/bin/sh
- This tells the system that the commands that follow are to be executed by the Bourne Shell.
- It is called shebang because the # symbol is called a hash and the ! Symbol is called a bang
- Save the script using .sh extension
- Make the script executale y
- $chmod +x filename.sh
- Execution - $./programname.sh

Variable Name

- The name of a variable can contain only letters (a to z or A to Z), numbers(0 t0 9) or underscore character(_).
- Valid variable names _ALI, TOKEN_A, VAR_1
- Invalid varibale names _var, -var, var-var

Accessing Values

- To access the values stored in a variable, prefix its name with the dollar sign ($)
- Example:

```
#!bin/sh
NAME = “Anu”
echo $NAME
```

User Input

- To read a variable

```
read var_name
```

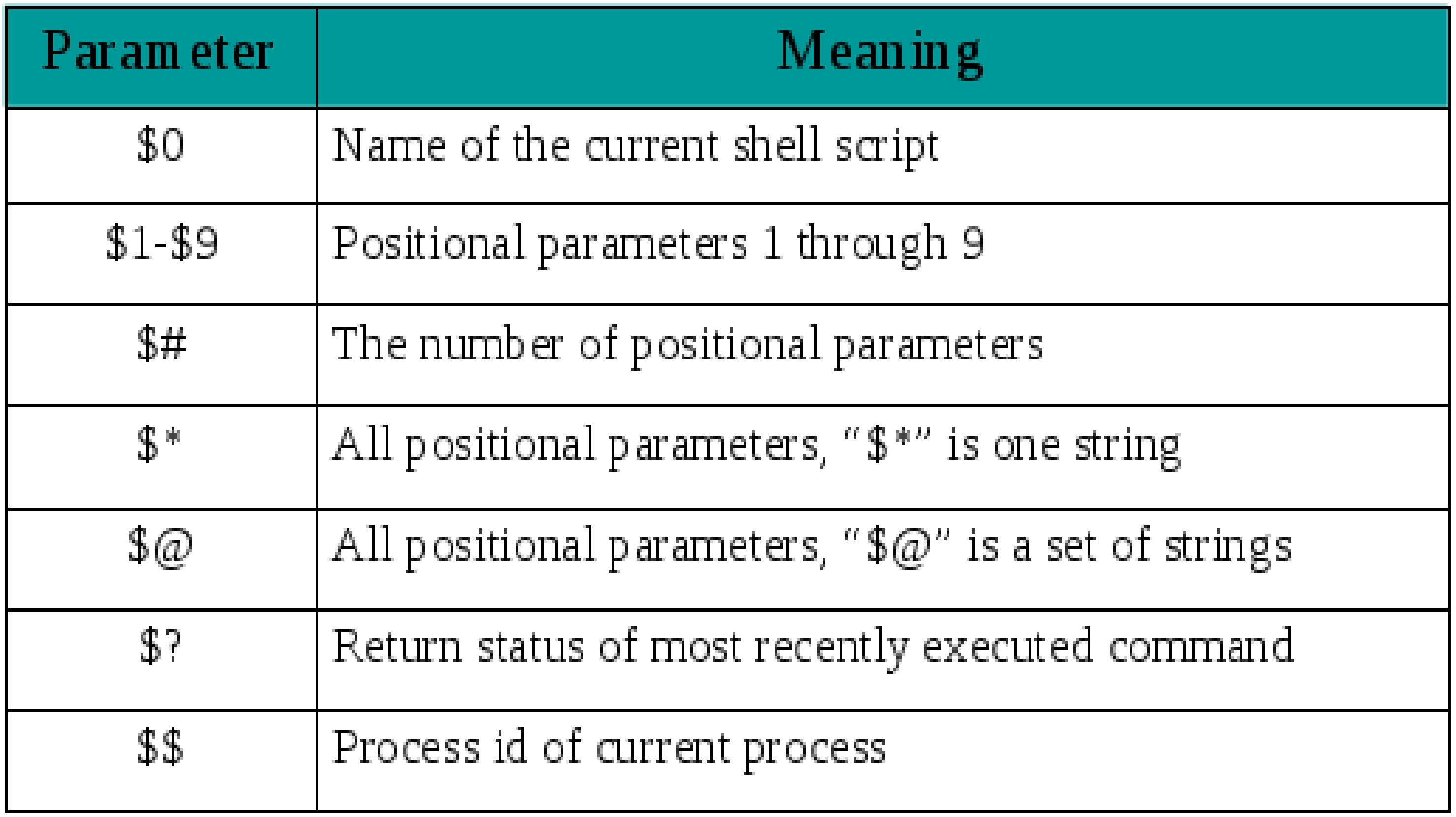
- You can print the variable using echo.

```
echo $var_name
```

- Let us write a small script.

```
#! / b i n / sh
read -p " enter your name : " first last
echo " F i r s t name : $ f i r s t "
echo " La s t name : $ l a s t "
```

Special Shell Variables



Parameter	Meaning
$0	Name of the current shell script
$1-$9	Positional parameters 1 through 9
$#	The number of positional parameters
$*	All positional parameters, "$*" is one string
$@	All positional parameters, "$@" is a set of strings
$?	Return status of most recently executed command
$$	Process id of current process

Experimenting with special shell variables

```
#! / b i n / bash

  echo " Name o f your s c r i p t i s $0"
  echo " arguement s  entered on command line $
*"
  echo " first arguement  $1"
  echo " Second arguement  $2“
  echo " You entered $# argument s "
  echo " process id is $$"
```

Control structures

Basic if statement

```
if [ <some test> ]
then
<commands>
fi
```

Example: Test whether a number input from command line is greater than 100

```
#!/bin/bash

if [ $1 -gt 100 ]
then
echo Hey that's a large number.

fi
```

test Command

test command is very commonly used with if in shell script.
test command can be used perform a variety of test on the system. It can be written in two ways as shown below.
Note that you need proper spacing around the square brackets.

```
test <EXPRESSION>

[ <EXPRESSION> ]
```

test command evaluates the expression and returns true or false.

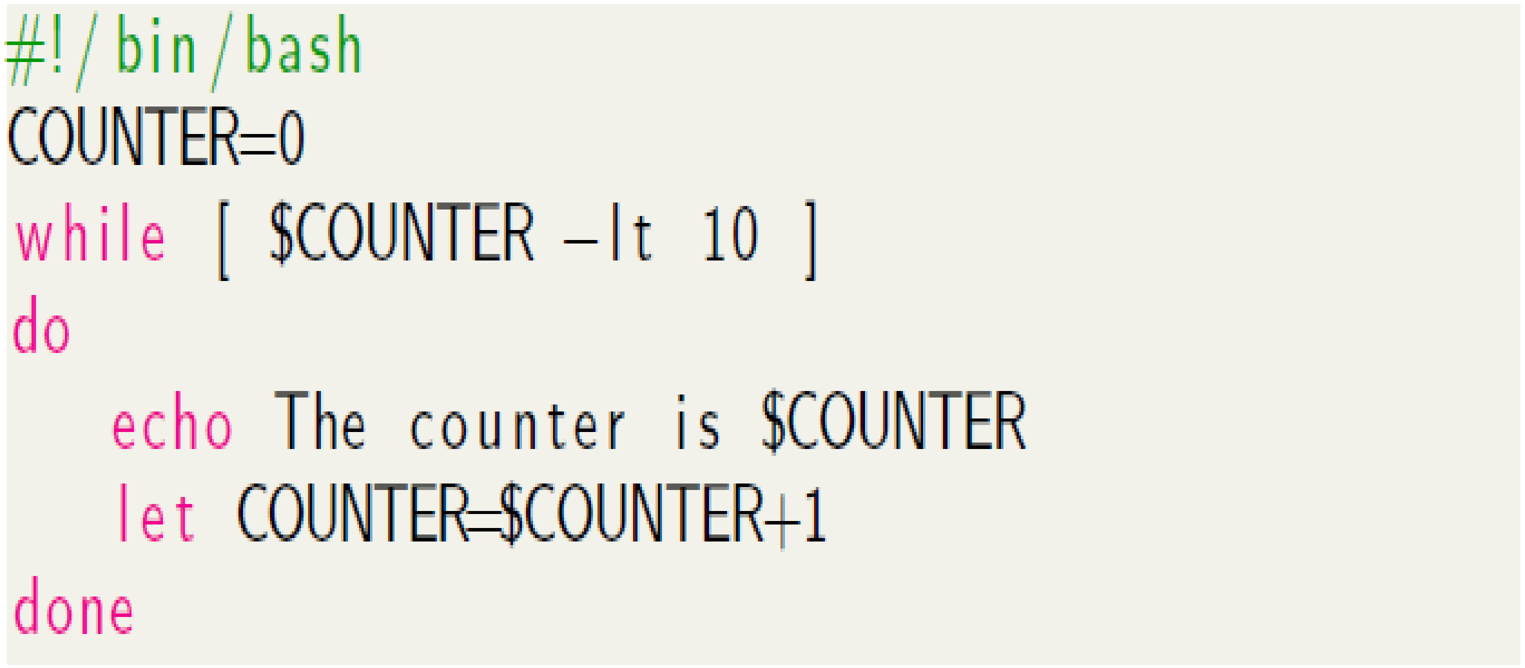
If elif

```bash
#!/bin/bash

read -p "Enter Income Amount: " Income
read -p "Enter Expenses Amount: " Expense

let Net=$Income-$Expense

if [ "$Net" -eq "0" ]; then
   echo "Income and Expenses are equal - breakeven."
elif [ "$Net" -gt "0" ]; then
   echo "Profit of: " $Net
else
   echo "Loss of: " $Net
fi
```

Case statement

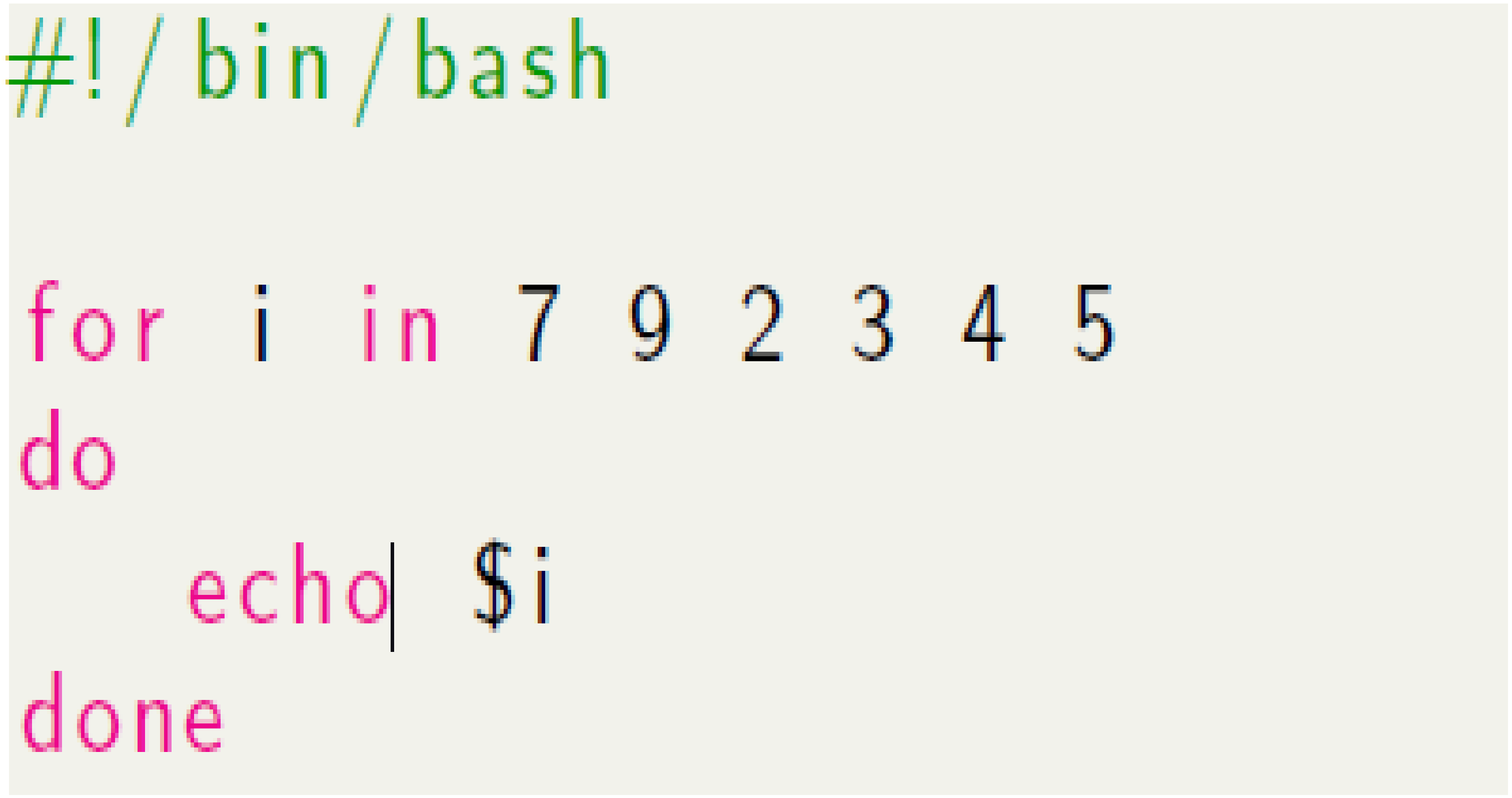
```
#!/bin/bash
echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter q to quit"

read -p "Enter your choice: " reply

case $reply in
  Y|YES) echo "Displaying all  files"
         ls -a ;;
  N|NO)  echo "Display all non-hidden files..."
         ls ;;
  Q)     exit 0 ;;

  *) echo "Invalid choice!"; exit 1 ;;
esac
```

While Loop



```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
    echo The counter is $COUNTER
    let COUNTER=$COUNTER+1
done
```

for Loop



```
#!/bin/bash

for i in 7 9 2 3 4 5
do
   echo $i
done
```