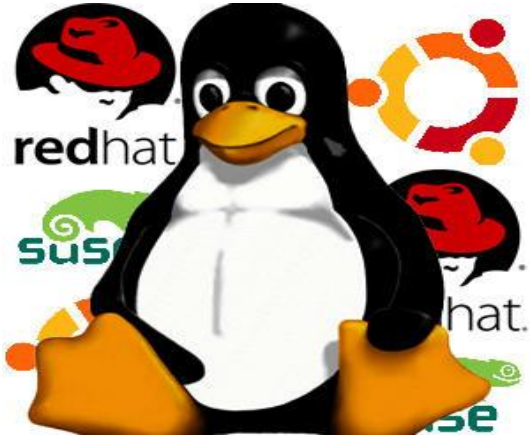
Unix Shell Scripting



By
Satish Mongam
DB2 DBA
IBM Certified Database Associate
smongam@miraclesoft.com

Certified for
IBM Information Management
software

MIRACLE MSS
SOFTWARE SYSTEMS, INC.

Agenda

- Shell Scripting Basics
- Shell Programming Features
- Shell Inputs & Outputs
- Variables
- Comments & White Spaces
- Quoting
- Operators
- Flow Control
- Loops

Shell Scripting Basics

- **SHELL:** It is a special program which provides you to interact with a UNIX-Based systems. It gathers input from the user executes it and then displays output.
- **SCRIPTS:** Stores Shell Commands in a file and execute the file as a program.
- **Shell Scripts:**
  - UNIX shell scripts are text files that contain sequences of UNIX commands.
  - Like high-level source files, a programmer creates shell scripts with a text editor .
  - Shell scripts do not have to be converted into machine language by a compiler. This is because the UNIX shell acts as an interpreter when reading script files.

Shell Programming Features

- Shell Inputs & Outputs
- Variables
- Comments & White Spaces
- Quoting
- Operators
- Flow Control
- Loops

1. Shell Inputs & Outputs

- **read** Command : Read command waits for the user to enter a value.

Eg : **read SID**

- **echo** Command : The echo command is mostly used for printing strings. It is also used to access the value stored in a variable, prefix its name with the dollar sign ($).

Eg : **echo $SID**

2. Variables

- Variables are set using (=) sign.

Eg: **ORACLE_SID=oss**

- Variables and their contents are case sensitive. So the variable **ORACLE_SID** is different from the variable **oracle_sid**.
- Shell variables are un-typed and it treats all the values as text only.
- Types Of Variables:
  - Environment Variables
  - Shell Variables
  - Special Variables
  - Array Variables

Variable Naming

- Variables should have meaningful names.
- Variable names do not need to be short.
- All UPPER CASE typically indicates an environment variable.
- Local (script) variables are conventionally all lover case.
- Underscores (_) are the best for separating words in the variable names.

Environment Variables

- An environment variable is a variable that is available to any child process of the shell.
- Some programs need environment variables in order to function correctly.
- Exporting Variables:
  - Syntax : **name=value ; export name** (OR)

    export name=value
  - Example : **SID=oss ; export SID** (OR)

    export SID=oss
- Deleting Variables :
  - Syntax: **unset name**
  - Eg: **unset SID**

Variable Scope and Usage

- Variables will be available within the script or shell session which sets them.
- By exporting variables they can be made available to subsequently called scripts.
- Exporting is not necessary when variables will only be used within the current script.
- The dollar sign ($) is used to retrieve the contents of a variable.

$ echo $ORACLE_SID
OSS

Shell Variables

- The shell provides with the capability to define variables and assign them values.
- A variable name is a sequence of characters beginning a letter or an underscore.
- There is no type associated with a shell variable. Every value that is assigned to a variable is treated as a string of characters by the shell.
  - Syntax : **variable_name=value**
  - Examples: **fruit=apple**

  name=”satish mongam”

  num1=23

  fruit_name=banana

NOTE : There must be no spaces around the “ = “ sign.

Special Variables (6)

- **?** - The previous command's exit status.
- $ - The PID of the current shell process.
- ! - The PID of the last command that was run in the background.
- 0 - The filename of the current script.
- (1-9) - The first through ninth command-line arguments given when the current script was invoked: $1 is the value of the first command-line argument, $2 the value of the second, and so forth.
- (_) - The last argument given to the most recently invoked command before this one.

Array Variables

- Arrays provide a method of grouping a set of variables. There are 3 methods to define an array variable.
- **Method One:** **name[index]=value**

Example:

$ FRUIT[0]=apple

$ FRUIT[1]=banana

$ FRUIT[2]=orangeComplex

- **Method Two:** **name=(value1 ... valuen)**

Example : **$ band=(derri terry mike)**

is equivalent to the following commands:

$ band[0]=derri

$ band[1]=terry

$ band[2]=mike

- **Method Three:**

Example : **myarray=([0]=derri [2]=mike [1]=terry)**

3. Comments & Whitespace

- Any thing appearing after a pound or hash symbol (#) on a line will be ignored.
- Terminate comment by pressing enter key.
- Adding comments can aid troubleshooting and future editing of the script.
- Blank lines are ignored when a script is executed.
- Blank lines and other whitespaces (tabs, spaces) can be used to improve script readability.

A Basic Script

```
#!/bin/bash

echo "The current database is $ORACLE_SID"

echo "The current running processes for
  $ORACLE_SID are"
```

This first line indicates what interpreter to use when running this script

```
#!/bin/bash

echo "The current database is $ORACLE_SID"

echo "The current running processes for
  $ORACLE_SID are"
```

Whitespace is used to separate commands to improve readability.

MIRACLE MSS
SOFTWARE SYSTEMS, INC.

The Shebang (#!)

- The “Shebang” is a special comment. Since it is a comment it will not executed when the script is run.
- Instead before the script run, the shell calling the script will check for the **#!** pattern. If found it will invoke the script using that interpreter. If no **#!** is found most shells will use the current shell to run the script.
- Since the shells are installed in different locations on different systems you may have to alter the **#!** line.
- For example, the bash shell may be in /bin/bash, /usr/bin/bash or /usr/local/bin/bash.
- Setting the shell explicitly like this assures that the script will be run with the same interpreter regardless of who executes it or what their default shell may be.

4. Quoting

- Turning off the special meaning of a character is called quoting.
- It can be done three ways:
  - Using the backslash (\)
  - Using the single quote (')
  - Using the double quote (")

Quoting with Backslashes (\)

- Putting a backslash (\)in front of the a character to take away its special meaning, enabling you to display it as a literal character.

Example:

name=satish
echo $name
satish
echo \$name
$name

Using Double Quotes (“ “)

- Double quotes take away the special meaning of all characters except the:
  - ($) for parameter substitution
  - (`) Backquotes for command substitution
  - (\) Backslashes.

Example :

name=satish
echo “My name is $name”
echo “My name is \$name”
My name is satish
My name is $name

Using Single Quotes (' ')

- Any characters within single quotes are quoted just as if a backslash is in front of each character.
- All characters inside single quotes are interrupted with no special meaning .

Example :

name=satish
surname=mongam
echo “$name $surname”
echo '\$name \$surname'

Output:

satish mongam
\$name \$surname

Command Substitution

- A pair of **backquotes /backticks / grave accents** (` `) does command substitution .
- This is really useful – to take output of a command or a list of commands to a variable.

Example:

mydir=`pwd`

echo “My Present Working Directory Is $mydir”

Output:

/home/mes

5. Flow Control : The if Statement

The simplest flow control statement is the `if` statement.

```
$ age=29
$ if [ $age -lt 30 ]
> then
> echo "You're still under 30"
> fi
You're still under 30
```

The if Statement Cont ...

The simplest flow control statement is the `if` statement.

```
$ age=29
$ if [ $age -lt 30 ]
> then
> echo "You're still under 30"
> fi
You're still under 30
```

Note that the end of an if statement is indicated by the keyword fi

if, elseif and else

```
#!/bin/sh
age=39
if [ $age -lt 30 ]
  then
     echo "You're still under 30"
  elif [ $age -ge 30 -a $age -le 40 ]
     then
          echo "You're in your 30s"
  else
     echo "You're 40 or over"
fi
```

if, elseif and else - Cont ...

```
#!/bin/sh
age=39
if [ $age -lt 30 ]
  then
     echo "You're still under 30"
  elif [ $age -ge 30 -a $age -le 40 ]
     then
          echo "You're in your 30s"
  else
     echo "You're 40 or over"
fi
```

Initially this condition is checked and, if true, the code in the `then` section executed

if, elseif and else - Cont ...

```
#!/bin/sh
age=39
if [  $age -lt 30 ]
  then
     echo "You're still under 30"
  elif [  $age -ge 30 -a $age -le 40 ]
     then
          echo "You're in
  else
     echo "You're 40 or over"
fi
```

Only if the initial condition has failed will the elif be considered

if, elseif and else - Cont ...

```
#!/bin/sh
age=39
if [  $age -lt 30 ]
  then
     echo "You're still under 30"
  elif [  $age -ge 30 -a $age -le 40 ]
     then
          echo "You're in your 30s"
  else
     echo "You're 40 or over"
fi
```

Finally if the if condition and all elif conditions have failed the else, if present, will be executed

if, elseif and else - Cont ...

- Conditional statements can compare numbers or text
- An `if` statement will need to have a `then` and an `fi` to indicate the end of the statement
- An `if` statement can have one or more `elif` statements or may have none
- An `if` statement may have one `else` statement but may have no `else` statement

The Case Statement

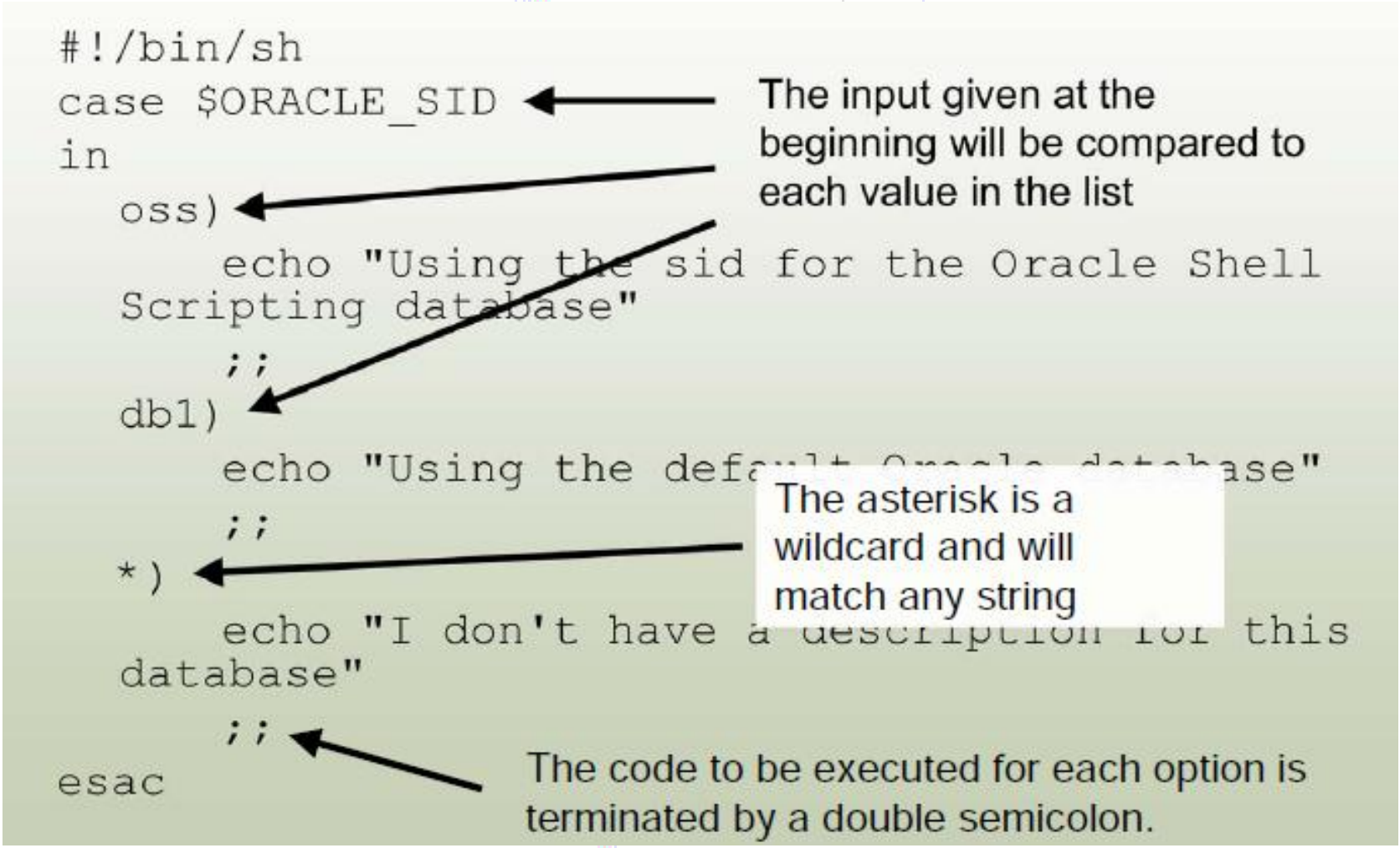
```
#!/bin/sh
case $ORACLE_SID
in
   oss)
       echo "Using the sid for the Oracle Shell
   Scripting database"
       ;;
   db1)
       echo "Using the default Oracle database"
       ;;
   *)
       echo "I don't have a description for this
   database"
       ;;
esac
```

The Case Statement cont ...

```
#!/bin/sh
case $ORACLE_SID
in
   oss)
       echo "Using the sid for
   Scripting database"
       ;;
   db1)
       echo "Using the default Oracle database"
       ;;
   *)
       echo "I don't have a description for this
   database"
       ;;
esac
```

The beginning of a case statement is indicated by the case keyword. The end is indicated by case spelled backwards

The Case Statement cont ...



MIRACLE MSS
SOFTWARE SYSTEMS, INC.

The Case Statement cont ...

- The code following the **first** matching option will be executed.
- If no match is found the script will continue on after the `esac` statement without executing any code.
- Some wildcards and regular expressions can be used.
- A `case` could be rewritten as a series of `elif` statements but a `case` is typically more easily understood.

6. Operators : Mathematical Operators

Mathematical Comparators

Comparator	Mathematic Equivalent	Evaluates to true if
-eq or =	=	the values on each side of the comparator are **equal**
-ne or !=	≠	the two values are **not equal**
-gt	>	the first value is **greater than** the second
-ge	≥	the first value is **greater than or equal to** the second
-lt	<	the first value is **less than** the second
-le	≤	the first value is **less than or equal to** the second

Logical Operators

Comparator Modifiers

Comparator	Evaluates to true if
-a	the expressions on **each side of the comparator are both true**
-o	**one or both of the expressions are true**
!	The following expression is **false**

Comparing Strings

```
$ if [ $ORACLE_SID = "oss" ]
> then
> echo "Using the sid for the Oracle Shell
  Scripting database"
> fi
Using the sid for the Oracle Shell Scripting
  database
```

File Comparison Operators

File Comparators

Comparator	Evaluates to true if
-nt	the file listed before is **newer than** the file listed after the comparator
-ot	the file listed before is **older than** the file listed after the comparator
-e	the file **exists**
-d	the file is a **directory**
-h	the file is a **symbolic link**
-s	the file is **not empty** (has a size greater than zero)
-r	the file is **readable**
-w	the file is **writable**

Checking Files

```
$ if [ -e
  $ORACLE_HOME/dbs/init$ORACLE_SID.ora ]
> then
> echo "An init file exists for the
  database $ORACLE_SID"
> fi
An init file exists for the database oss
```

Checking Multiple Files

```
$ if [ -e $ORACLE_HOME/dbs/init$ORACLE_SID.ora -a -e \
> $ORACLE_HOME/dbs/spfile$ORACLE_SID.ora ]
> then
> echo "We seem to have both an spfile and an init file"
> fi
We seem to have both an spfile and an init file
```

Arithmetic Operations With Shell Variables

- Since the shell variables treated as characters , a different mechanism is adopted to perform arithmetic operations on shell variables.
- Use ***expr*** utility to perform arithmetic operations.

```
#!/bin/bash
# Adding 2 numbers
num1=23
num2=30
total=`expr $num1 \+ $num2`
```

7. Loops : The while Loop

The `while` loop will repeat a chunk of code as long as the given condition is true.

```
#!/bin/sh
i=1
while [ $i -le 10 ]
do
  echo "The current value of i is $i"
  i=`expr $i + 1`
done
```

The while Loop cont ...

```
#!/bin/sh
i=1
while [ $i -le 10 ]
do
  echo "The current value of i is $i"
  i=`expr $i + 1`
done
```

Make sure your loop variable is initialized before the loop starts

Also makes sure that something will eventually cause the while condition to fail, otherwise you may end up in an infinite loop!

The for Loop

The for loop allows you to easily parse a set of values.

```
#!/bin/sh
count=0
for i in 2 4 6
do
  echo "i is $i"
  count=`expr $count + 1`
done
echo "The loop was executed $count times"
```

The for Loop cont ...

```
#!/bin/sh
count=0
for i in 2 4 6
do
  echo "i is $i"
  count=`expr $count + 1`
done
echo "The loop was executed $count
  times"
```

This for loop will be executed three times, once with i=2, once with i=4 and once with i=6

Breaking Out Of The Current Loop

The `break` statement will cause the shell to stop executing the current loop and continue on after its end.

```
#!/bin/sh
files=`ls`
count=0
for i in $files
do
   count=`expr $count + 1`
   if [ $count -gt 100 ]
   then
       echo "There are more than 100 files in the current
   directory"
       break
   fi
done
```

Naming Convention of Scripts

- Use full words. Descriptive names are very important.
- Separate words with underscores.
- Avoid using spaces or other unusual characters.
- There is no requirement for script names, but tipically they will end with **.sh** extension.

Calling a Script

- **source filename** – Needs no execute permission
- **sh filename** – Needs no execute permission
- **. filename** – Needs no execute permission
- **filename** – Needs execute permission

Example:

$sh sample.sh

Any Queries ...

MIRACLE MSS
SOFTWARE SYSTEMS, INC.

Thank
Q

MIRACLE MSS
SOFTWARE SYSTEMS, INC.