Looping Statements | Shell Script

Looping Statements in Shell Scripting: There are total 3 looping statements that can be used in bash programming

- `while` statement in Shell Script in Linux
- `for` statement in Shell Script in Linux
- `until` statement in Shell Script in Linux
- Examples of Looping Statements

To alter the flow of loop statements, two commands are used they are,

1. break
2. continue

Their descriptions and syntax are as follows:

`while` statement in Shell Script in Linux

Here the command is evaluated and based on the resulting loop will execute, if the command is raised to false then the loop will be terminated **that.**

```
#/bin/bash
while <condition>
do
   <command 1>
   <command 2>
   <etc>
done
```

Implementation of `While` Loop in Shell Script.

First, we create a file using a text editor in Linux. In this case, we are using `vim`editor.

```
vim looping.sh
```

You can replace “looping.sh” with the desired name.

Then we make our script executable using the `chmod` command in Linux.

```
chmod +x looping.sh
```

```
#/bin/bash

a=0

# It is less than operator
```

```
#Iterate the loop until a less than 10

while [ $a -lt 10 ]
do
# Print the values
echo $a
# increment the value
a=`expr $a + 1`
done
```

Output:

While Loop in Linux

Explanation:

- **#/bin/bash**: Specifies that the script should be interpreted using the Bash shell.
- **a=0**: Initializes a variable a with the value 0.
- **while [$a -lt 10]**: Initiates a while loop that continues as long as the value a is less than 10.
- **do**: Marks the beginning of the loop's body.
- **echo $a**: Prints the current value of a the console.
- **a=expr $a + 1"**: Increments the value of a by 1. The expr command is used for arithmetic expressions.
- **done**: Marks the end of the loop.

\`for\` statement in Shell Script in Linux

The for loop operates on lists of items. It repeats a set of commands for every item in a list. Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Syntax:

```
#/bin/bash
for <var> in <value1 value2 ... valuen>
do
   <command 1>
   <command 2>
   <etc>
done
```

Implementation of `for` Loop with `break` statement in Shell Script.

First, we create a file using a text editor in Linux. In this case, we are using `vim`editor.

```
vim for.sh
```

You can replace “for.sh” with the desired name.

Then we make our script executable using the `chmod` command in Linux.

```
chmod +x for.sh
```

```
#/bin/bash

#Start of for loop

for a in 1 2 3 4 5 6 7 8 9 10
do

# if a is equal to 5 break the loop
if [ $a == 5 ]
then
break
fi

# Print the value
echo “Iteration no $a”
done
```

Output:

Break statement in for Loop in linux

Explanation:

- #/bin/bash: Specifies that the script should be interpreted using the Bash shell.
- for a in 1 2 3 4 5 6 7 8 9 10: Initiates a for loop that iterates over the values 1 through 10, assigning each value to the variable a in each iteration.
- do: Marks the beginning of the loop’s body.
- if [$a == 5]: Checks if the current value a is equal to 5.
    - then: Marks the beginning of the conditional block to be executed if the condition is true.
        - break: Exits the loop if the condition is true.
    - fi: Marks the end of the conditional block.
- echo "Iteration no $a": Prints a message indicating the current iteration number.
- done: Marks the end of the loop.

The script sets up a for loop that iterates over the values 1 through 10. During each iteration, it checks if the value a is equal to 5. If it is, the loop is exited using the break statement. Otherwise, it prints a message indicating the current iteration number. The loop continues until it completes all iterations or until it encounters a break statement.

Implementation of `for` Loop with `continue` statement in Shell Script.

First, we create a file using a text editor in Linux. In this case, we are using `vim`editor.

vim for_continue.sh

You can replace “for_continue.sh” with the desired name.

Then we make our script executable using the `chmod` command in Linux.

chmod +x for_continue.sh

#/bin/bash

for a in 1 2 3 4 5 6 7 8 9 10
do

# if a = 5 then continue the loop and
# don’t move to line 8

if [$a == 5]
then
continue

```
fi
echo “Iteration no $a”
done
```

Output:

continue statement in for loop in Linux

Explanation:

- **#/bin/bash**: Specifies that the script should be interpreted using the Bash shell.
- **for a in 1 2 3 4 5 6 7 8 9 10**: Initiates a for loop that iterates over the values 1 through 10, assigning each value to the variable a in each iteration.
- **do**: Marks the beginning of the loop’s body.
- **if [$a == 5]**: Checks if the current value a is equal to 5.
    - **then**: Marks the beginning of the conditional block to be executed if the condition is true.
        - **continue**: Skips the rest of the loop’s body and goes to the next iteration if the condition is true.
    - **fi**: Marks the end of the conditional block.
- **echo "Iteration no $a"**: Prints a message indicating the current iteration number. This line is skipped if a is equal to 5 due to the continue statement.
- **done**: Marks the end of the loop.

The script sets up a for loop that iterates over the values 1 through 10. During each iteration, it checks if the value a is equal to 5. If it is, the loop continues to the next iteration without executing the remaining statements in the loop’s body. Otherwise, it prints a message indicating the current iteration number. The loop continues until it completes all iterations.

`until` statement in Shell Script in Linux

The until loop is executed as many times as the condition/command evaluates to false. The loop terminates when the condition/command becomes true.

Syntax:

```
#/bin/bash
until <condition>
do
   <command 1>
   <command 2>
   <etc>
done
```

Implementing `until` Loop in Shell Scrip

First, we create a file using a text editor in Linux. In this case, we are using `vim`editor.

```
vim until.sh
```

You can replace “until. sh” with the desired name.

Then we make our script executable using the `chmod` command in Linux.

```
chmod +x until.sh
```

```
#/bin/bash

a=0

# -gt is greater than operator
#Iterate the loop until a is greater than 10

until [ $a -gt 10 ]
do

# Print the values
echo $a

# increment the value
a=`expr $a + 1`
done
```

Output:

Until loop in Linux

Explanation:

- **#/bin/bash**: Specifies that the script should be interpreted using the Bash shell.
- **a=0**: Initializes a variable a with the value 0.
- **until [$a -gt 10]**: Initiates a until loop that continues as long as the value a is not greater than 10.
- **do**: Marks the beginning of the loop's body.
- **echo $a**: Prints the current value of a the console.
- **a=expr $a + 1"**: Increments the value of a by 1. The expr command is used for arithmetic expressions.
- **done**: Marks the end of the loop.

Note: Shell scripting is a case-sensitive language, which means proper syntax has to be followed while writing the scripts.

Examples of Looping Statements

1. Iterating Over Colors Using a For Loop

First, we create a file using a text editor in Linux. In this case, we are using `vim`editor.

vim color.sh

You can replace "color.sh" with the desired name.

Then we make our script executable using `chmod` command in Linux.

chmod +x color.sh

#/bin/bash

```
COLORS=”red green blue”

# the for loop continues until it reads all the values from the COLORS

for COLOR in $COLORS
do
echo “COLOR: $COLOR”
done
```

Output:

For until in Linux

Explanation:

- **Initialization of Colors:**
    - COLORS="red green blue": Initializes a variable named COLORS with a space-separated list of color values (“red”, “green”, and “blue”).
- **For Loop Iteration:**
    - for COLOR in $COLORS: Initiates a for loop that iterates over each value in the COLORS variable.
        - **Loop Body:**
            - echo "COLOR: $COLOR": Prints a message for each color, displaying the current color value.
    - The loop continues until it processes all the values present in the COLORS variable.

This example demonstrates a simple for loop in Bash, iterating over a list of colors stored in the COLORS variable. The loop prints a message for each color, indicating the current color being processed. The loop iterates until all color values are exhausted.

2. Creating an Infinite Loop with “while true” in Shell Script

First we create a file using a text editor in Linux. In this case we are using `vim`editor.

```
vim infinite.sh
```

You can replace “infinite.sh” with desired name.

Then we make our script executable using `chmod` command in Linux.

```
chmod +x infinite.sh
```

```
#/bin/bash

while true
do

# Command to be executed
# sleep 1 indicates it sleeps for 1 sec
echo “Hi, I am infinity loop”
sleep 1
done
```

Output:

infinite loop in linux

Explanation:

Infinite Loop Structure:

- while true: Initiates a while loop that continues indefinitely as the condition true is always true.
    - **Loop Body:**
        - echo "Hi, I am infinity loop": Prints a message indicating that the script is in an infinite loop.
        - sleep 1: Pauses the execution of the loop for 1 second before the next iteration.
- The loop continues indefinitely, executing the commands within its body repeatedly.

This example showcases the creation of an infinite loop using the while true construct in Bash. The loop continuously prints a message indicating its status as an infinite loop and includes a sleep 1 command, causing a one-second delay between iterations. Infinite loops are often used for processes that need to run continuously until manually interrupted.

3. Interactive Name Confirmation Loop

First we create a file using a text editor in Linux. In this case we are using `vim`editor.

```
vim name.sh
```

You can replace “name.sh” with desired name.

Then we make our script executable using `chmod` command in Linux.

```
chmod +x name.sh
```

```
#/bin/bash

CORRECT=n
while [ “$CORRECT” == “n” ]
do

# loop discontinues when you enter y i.e., when your name is correct
# -p stands for prompt asking for the input

read -p “Enter your name:” NAME
read -p “Is ${NAME} correct? ” CORRECT
done
```

Output:

Interactive script in Linux

Explanation:

- **Initialization:**
  - CORRECT=n: Initializes a variable named CORRECT with the value “n”. This variable is used to control the loop.
- **Interactive Loop:**
  - while ["$CORRECT" == "n"]: Initiates a while loop that continues as long as the value of CORRECT is equal to “n”.
    - **Loop Body:**
      - read -p "Enter your name:" NAME: Prompts the user to enter their name and stores the input in the NAME variable.

    - read -p "Is ${NAME} correct? " CORRECT: Asks the user to confirm if the entered name is correct and updates the CORRECT variable accordingly.
  - The loop continues until the user enters “y” (indicating the correct name).

This example demonstrates an interactive loop that prompts the user to enter their name and then asks for confirmation. The loop continues until the user confirms that the entered name is correct by inputting “y”. The loop utilizes the `**read**` command for user input and updates the `**CORRECT**` variable to control the loop flow.

Conclusion

In this article we discussed looping statements in Bash scripting, covering while, for, and until loops. It introduces the use of break and continue statements to modify loop behavior. Practical examples illustrate the implementation of loops, including iterating over color values, creating infinite loops, and building an interactive loop for user input validation. The guide offers a concise yet informative resource for understanding and utilizing looping constructs in Bash scripting