ShellLoopType

April 15, 2024

1 Loops In Shell Script?

Shell scripting supports several types of loops that can be used to perform repetitive tasks. Here are the main ones:

1. **For Loop**: The `for` loop is used to perform a block of code a set number of times.

```
for i in 1 2 3 4 5; do
    echo "Looping number $i"
done
```

2. **While Loop**: The `while` loop is used to perform a block of code as long as a condition is true.

```
count=1
while [ $count -le 5 ]; do
    echo "Count: $count"
    count=$((count+1))
done
```

3. **Until Loop**: The `until` loop is used to perform a block of code until a condition is true.

```
count=1
until [ $count -gt 5 ]; do
    echo "Count: $count"
    count=$((count+1))
done
```

4. **Select Loop**: The `select` loop is used to generate a menu from a list of items.

```
select option in "Option 1" "Option 2" "Option 3"; do
    echo "You selected $option"
    break
done
```

5. **Nested Loop**: A loop inside another loop is called a nested loop.

```
for i in 1 2; do
    for j in 3 4; do
        echo "i=$i, j=$j"
    done
done
```

Note: Always use spaces around the brackets in the test conditions. For example, use `[ $count -le 5 ]` instead of `[$count -le 5]`.

2 break

In shell scripting, you can use the `break` statement to exit a loop prematurely. Here's an example:

[ ]:
```
for i in 1 2 3 4 5; do
    if [ $i -eq 3 ]; then
        break
    fi
    echo "Number: $i"
done
# Outputs: Number: 1
#          Number: 2
```

In this example, the `for` loop would normally iterate over the numbers 1 through 5. However, the `break` statement causes the loop to exit as soon as `$i` equals 3. As a result, the numbers 1 and 2 are printed, but the numbers 3, 4, and 5 are not.

You can also specify the level of loop to break out of by providing a number to the `break` statement. For example, `break 2` would break out of two levels of loops. This is useful in nested loop situations.

3 Continue statement?

In shell scripting, you can use the `continue` statement to skip the rest of the current loop iteration and continue with the next one. Here's an example:

[ ]:
```
for i in 1 2 3 4 5; do
    if [ $i -eq 3 ]; then
        continue
    fi
    echo "Number: $i"
done
# Outputs: Number: 1
#          Number: 2
#          Number: 4
#          Number: 5
```

In this example, the `for` loop iterates over the numbers 1 through 5. However, the `continue` statement causes the loop to skip the rest of the current iteration as soon as `$i` equals 3. As a result, the numbers 1, 2, 4, and 5 are printed, but the number 3 is not.

Just like the `break` statement, you can also specify the level of loop to continue to by providing a number to the `continue` statement. For example, `continue 2` would skip to the next iteration of the second enclosing loop. This is useful in nested loop situations.

4 The infinite Loop?

An infinite loop is a loop that never ends. In other words, the loop continues to execute indefinitely because the condition for its exit is never met.

In shell scripting, an infinite loop can be created in several ways. Here are a few examples:

1. **Using a `while` loop with a condition that is always true**:

```
while true; do
    echo "This is an infinite loop. Press Ctrl+C to stop."
done
```

2. **Using an `until` loop with a condition that is always false**:

```
until false; do
    echo "This is an infinite loop. Press Ctrl+C to stop."
done
```

3. **Using a `for` loop with an endless sequence**:

```
for (( ; ; )); do
    echo "This is an infinite loop. Press Ctrl+C to stop."
done
```

In all these examples, the loop will continue to execute until it is forcibly stopped. The most common way to stop an infinite loop is to send an interrupt signal to the shell process by pressing `Ctrl+C`.

Infinite loops are generally used when a script needs to keep running until it's manually stopped, or when it's waiting for a certain condition to be met that's changed by another process or event. However, they should be used with caution, as they can consume a lot of system resources and potentially cause your system to become unresponsive.

5 Thank You!