Repeating code with shell loops

In Bash scripting, **loops** are used to repeat a block of code multiple times. There are three primary types of loops in Bash:

1. **for loop**: Repeats a block of code a specific number of times or iterates over a list.
2. **while loop**: Repeats a block of code as long as a condition is true.
3. **until loop**: Repeats a block of code until a condition becomes true.

1. for Loop

The for loop iterates over a list of items or a range of numbers. It's useful when you know in advance how many times you need to repeat the block of code.

Syntax:

```
for var in list
do
  # Commands to execute
done
```

Example with a list of values:

```
#!/bin/bash
# Loop through a list of items

for item in apple banana cherry
do
  echo "Fruit: $item"
done
```

Output

```
Fruit: apple
Fruit: banana
Fruit: cherry
```

Example with a range of numbers:

```
#!/bin/bash
# Loop through numbers from 1 to 5

for i in {1..5}
do
```

```
  echo "Number: $i"
done
```

Output

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

2. while Loop

The **while** loop repeatedly executes a block of code as long as a specified condition evaluates to true.

Syntax:

```
while [ condition ]
do
  # Commands to execute
done
```

Example

```
#!/bin/bash
# Print numbers from 1 to 5 using a while loop

i=1
while [ $i -le 5 ]
do
  echo "Number: $i"
  ((i++))
done
```

Output

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

3. until Loop

The **until** loop repeatedly executes a block of code until a specified condition becomes true. It is the opposite of the `while` loop.

Syntax:

```
until [ condition ]
do
  # Commands to execute
done
```

Example

```
#!/bin/bash
# Print numbers from 1 to 5 using an until loop

i=1
until [ $i -gt 5 ]
do
  echo "Number: $i"
  ((i++))
done
```

Output

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

4. `for` Loop with C-style Syntax

You can also write `for` loops in a more traditional C-style syntax, useful when you need more complex loop control (such as initializing, checking a condition, and incrementing in a single line).

Syntax:

```
for ((i=0; i<5; i++))
do
  # Commands to execute
done
```

Example

```
#!/bin/bash
```

```
# C-style for loop example

for ((i=1; i<=5; i++))
do
  echo "Number: $i"
done
```

Output

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

5. Using Loops with break and continue

- break: Exits the loop early.
- continue: Skips the current iteration and continues with the next one.

Example with break and continue:

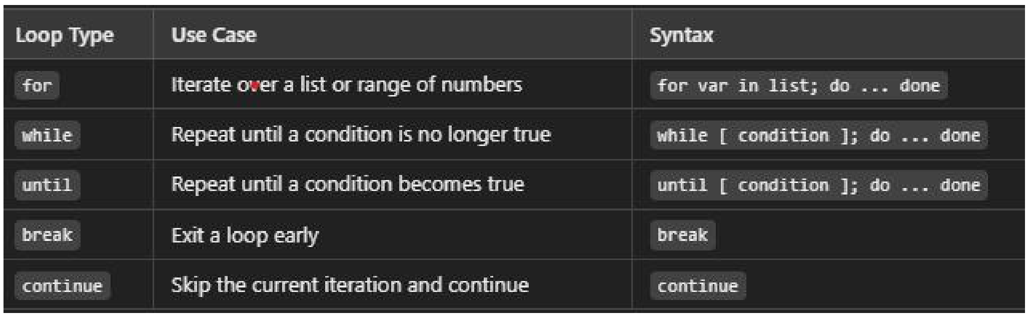
```
#!/bin/bash
# Loop with break and continue

for i in {1..10}
do
  if [ $i -eq 5 ]; then
    continue  # Skip the number 5
  fi
  if [ $i -eq 8 ]; then
    break  # Stop the loop when reaching 8
  fi
  echo "Number: $i"
Done
```

Output

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 6
Number: 7
```

The loop skips 5 due to the continue statement and stops at 8 because of the break statement.

Summary



Loop Type	Use Case	Syntax
`for`	Iterate over a list or range of numbers	`for var in list; do ... done`
`while`	Repeat until a condition is no longer true	`while [ condition ]; do ... done`
`until`	Repeat until a condition becomes true	`until [ condition ]; do ... done`
`break`	Exit a loop early	`break`
`continue`	Skip the current iteration and continue	`continue`