Leiden University
Medical Center

Scripting for Life Science Researchers

Advanced scripting II

Jeroen Laros

Outline

Loops

Two types

Bash has two types of loops:

- `for`.
- `while`.

`for` is used to loop over arrays.
`while` is used for conditional looping.

for-loops

General structure of a for-loop:

```
for variable in array; do
  body
done
```

The variable assumes all values in the array consecutively, starting with the one with the lowest index.

For every value, the body is executed.

This variable is usually used inside the body of the loop.

for-loops

```
1  $ for word in ${name[*]}; do
2  >   echo "value: ${word}"
3  > done
4  value: John
5  value: Jane
6  value: Johnny
7  value: Jane
```

Listing 1: Loop over an array.

for-loops

```
1   $ for word in 'one' 'two' 'three'; do
2   >   echo "value: ${word}"
3   > done
4   value: one
5   value: two
6   value: three
7
8   $ for word in {0..2}; do
9   >   echo "value: ${word}"
10  > done
11  value: 0
12  value: 1
13  value: 2
```

Listing 2: Loop over fixed sized arrays.

for-loops

We can also loop over the indices with `${!array[*]}`.

```
1   $ for value in ${!name[*]}; do
2   >   echo "value ${value}: ${name[${value}]}"
3   > done
4   value 0: John
5   value 1: Jane
6   value 2: Johnny
7   value 3: Jane
```

Listing 3: Print all indices and values.

Looping through associative arrays

With the following code snippets we can print all keys and their associated values.

```
$ for key in ${!age[*]}; do
>   echo "${key}: ${age[${key}]}"
> done
Jane: 32
John: 29
Johnny: 2
```

Listing 4: Print all values in an associative array.

Exercise

Run FizzBuzz for the first 20 positive integers.

while-loops

General structure of a while-loop:

```
while expression; do
  body
done
```

The body will be executed until the expression evaluates to false.

while-loops

The expression is usually manipulated from within the body.

```
1   i=0
2   while [ ${i} -lt 10 ]; do
3     sleep 1
4     i=$((${i} + 1))
5   done
```

Listing 5: Sleep for 10 seconds.

```
1   alarm=$(($(date +%s) + 10))
2   while [ $(date +%s) -lt ${alarm} ]; do
3     sleep 1
4   done
```

Listing 6: Sleep for 10 more seconds.

while-loops

A more complicated example.

```
1   $ value=8
2
3   $ divisions=0
4   $ while [ ${value} -ne 1 ]; do
5   >   value=$((${value} / 2))
6   >   divisions=$((${divisions} + 1))
7   > done
8
9   $ echo ${divisions}
10  3
```

Listing 7: Find the closest power of 2.

Structure of a function

General structure of a function:

```
name() {
  body
}
```

Parameters are used to pass information to the function.

Like the argument list, the parameters are numbered, i.e., `${1}`, `${2}`, etc.

Structure of a function

Example:

```
1   greetings() {
2     echo "Hello ${1}."
3   }
```

Listing 8: A function that greets.

```
1   $ greetings 'John'
2   Hello John.
```

Listing 9: Calling a function.

Local variables

By default, variables in functions are shared with the rest of the script.

```
1   $ inc() {
2   >   i=$((${i} + 2))
3   > }
4
5   $ i=10
6   $ inc
7   $ echo ${i}
8   11
```

Listing 10: The variable `i` is changed.

To avoid this (usually unwanted) behaviour, use `local` variables.

Local variables

```
1   $ inc() {
2   >   local i=$((${i} + 2))
3   > }
4
5   $ i=10
6   $ inc
7   $ echo ${i}
8   10
```

Listing 11: The variable `i` is not changed.

You can now reuse variables without having to worry about overwriting global variables.

The Collatz conjecture

Procedure:

- Start with any positive integer n.
- If n is even, divide by 2.
- Otherwise, multiply by 3 and add 1.

The conjecture is that no matter what value of n, the sequence will always reach 1.

Example: 12, 6, 3, 10, 5, 16, 8, 4, 2, 1.

Make a function that calculates the number of operations needed before reaching 1.

`https://en.wikipedia.org/wiki/Collatz_conjecture`

Leiden University
Medical Center

Acknowledgements

Jeroen Laros
Magnus Palmblad
Jonathan Vis
Mihai Lefter
Yassene Mohammed