For Loop

- What is a loop
- What is a brace expansion
- Loop control

TRAINING

CENTER

epam

Loops

Loops are special constructions in Bash that help us to repeat some piece of code based on some condition or data structure.

Loop	Description
For	Repeats code block for every item in passed variable.
While	Repeats code block while the conditional is true.
Until	Repeats code block until the conditional is true.

For Loop

For loop is used to iterate over something iterable.
For loop can either iterate over arrays, or special things called iterator, that is an incremented or decremented integer.

for var [in array]; do commands; done

```
bash-3.2$ array=( one two three )
bash-3.2$ for i in "${array[@]}"; do echo $i; done
one
two
three
bash-3.2$
```

For each element in array, var is set to that element and can be used in commands.

for ((var=n; conditional; calculation)); do commands; done

```
bash-3.2$ for ((i=1;i<=3;i++));do echo $i; done
1
2
3
bash-3.2$
```

First, var is set to a certain value, then a conditional is set, and calculation dictates how the variable is changed between iterations. The execution will continue while the conditional is true.

Brace Expansion { }

Bash has a mechanism for generating arbitrary array of strings called **brace expansion { }**. It is widely used in For loop to iterate over some generated values or even ranges of values.

```
bash-3.2$ for i in {foo,bar,baz}; do echo $i; done
foo
bar
baz
```

generating array via brace expansion

```
bash-3.2$ for i in {1..3}; do echo $i; done
1
2
3
```

generating numbers range

```
bash-3.2$ for i in fo{o..q}; do echo $i; done
foo
fop
foq
```

generating array with part of a string being variable

```
bash-3.2$ for i in {e..c}; do echo $i; done
e
d
c
```

generating reverse char range

```
bash-3.2$ for i in /var/log/{fo{o..p},ba{r{1..2},s{3..4}}}.log; do echo $i; done
/var/log/foo.log
/var/log/fop.log
/var/log/bar1.log
/var/log/bar2.log
/var/log/bas3.log
/var/log/bas4.log
```

more generative example

Loop Control

While loop is being executed, you may need to skip an iteration, or even exit the loop completely. For these purposes there're 2 commands exist: **break** and **continue**.

```
#!/bin/bash

for i in {1..1000}; do
  if [ $i -eq 5 ]; then
    echo "skipping the 5"
    continue
  fi
  if [ $i -eq 7 ]; then
    echo "exiting loop at 7"
    break
  fi
  echo "[[[   number $i ]]]"
done
echo "Good bye!"
```

example code

```
bash-3.2$ ./example1.sh
[[[   number 1 ]]]
[[[   number 2 ]]]
[[[   number 3 ]]]
[[[   number 4 ]]]
skipping the 5
[[[   number 6 ]]]
exiting loop at 7
Good bye!
```

example result

continue – stops code block execution and moves the loop to the next iteration.

break – stops code block execution and exits the loop.

For Loop Example

```
bash-3.2$ cat example.sh
#!/bin/bash
WAIT_TIME=${1:-5}

docker run --name influx -p 8086:8086 -d influxdb:1.7

for i in {5..1}; do
  sleep $WAIT_TIME
  curl localhost:8086/ping
  if [ $? -eq 0 ]; then
    echo "InfluxDB is up and running!"
    exit 0
  fi
  echo "$((i - 1)) attempts left,.."
done

echo "I am tired of waiting, killing InfluxDB container..."
docker rm -f influx
exit 1
bash-3.2$
```

example code

```
bash-3.2$ ./example.sh 0
e3593ed3b705219481ad9cb547e8339894d06e32fd62d06098d042ecd2b421f8
curl: (52) Empty reply from server
4 attempts left,..
curl: (52) Empty reply from server
3 attempts left,..
curl: (52) Empty reply from server
2 attempts left,..
curl: (52) Empty reply from server
1 attempts left,..
curl: (52) Empty reply from server
0 attempts left,..
I am tired of waiting, killing InfluxDB container...
influx
bash-3.2$ ./example.sh 1
77ca787b10c883f33cdcffa26124e4120d065f7e98cd2214a73ff7a8a22c201c
curl: (52) Empty reply from server
4 attempts left,..
curl: (52) Empty reply from server
3 attempts left,..
curl: (52) Empty reply from server
2 attempts left,..
InfluxDB is up and running!
```

example result

Thanks for watching!