C2110 *UNIX and Programming*

8th Lesson

Bash - Control Structures (Conditions, Cycles)

Petr Kulhánek

kulhanek@chemi.muni.cz

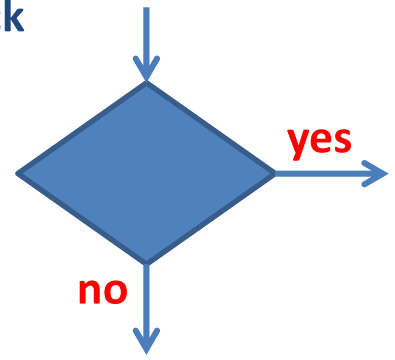
National Centre for Biomolecular Research, Faculty of Science,
Masaryk University, Kamenice 5, CZ-62500 Brno

Contents

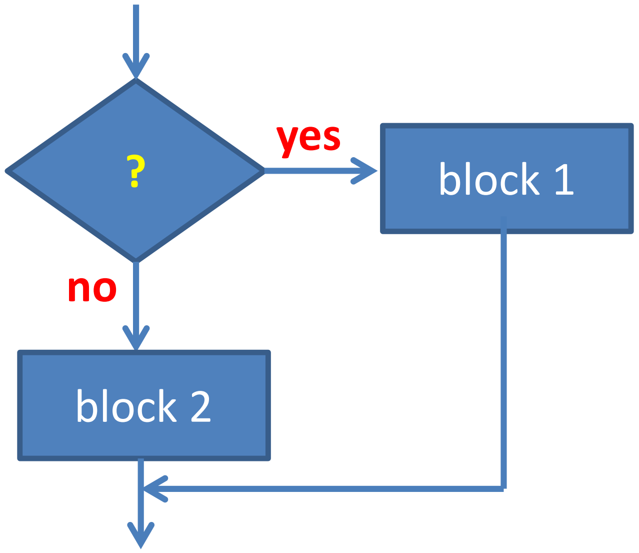
- **Decision making block**
  - **conditions, loops**
- **Return value of processes**
  - **command exit**
- **Command test**
  - **comparison operators, logical operators, simplified script**

Decision making block

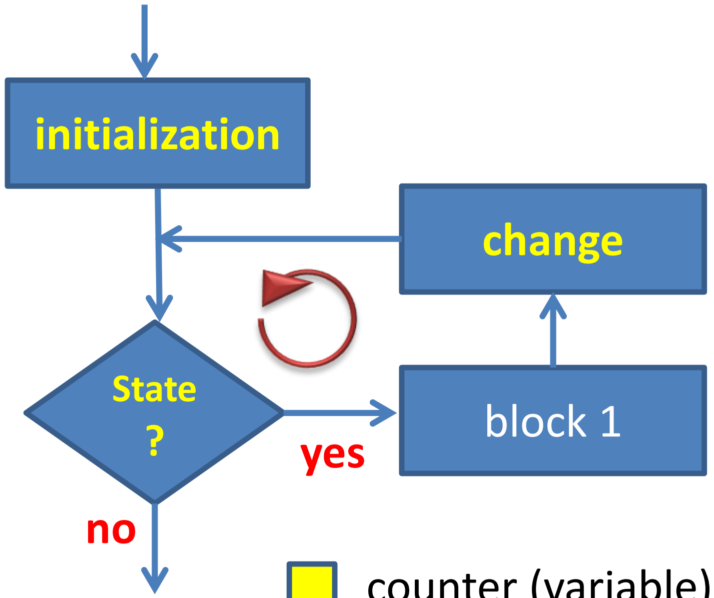
Typical use of decision-making block



Conditional execution of a block (conditions)



Cyclic execution of a block (loops)



🟨 counter (variable)

Conditions

```
if  command1
        then
        command2
                ...
        fi
```

```
if  command1
        then
        command2
                ...
        else
        command3
                ...
        fi
```

If **command1** exits with return value **0**, **command2** is executed, otherwise **command3** is executed.

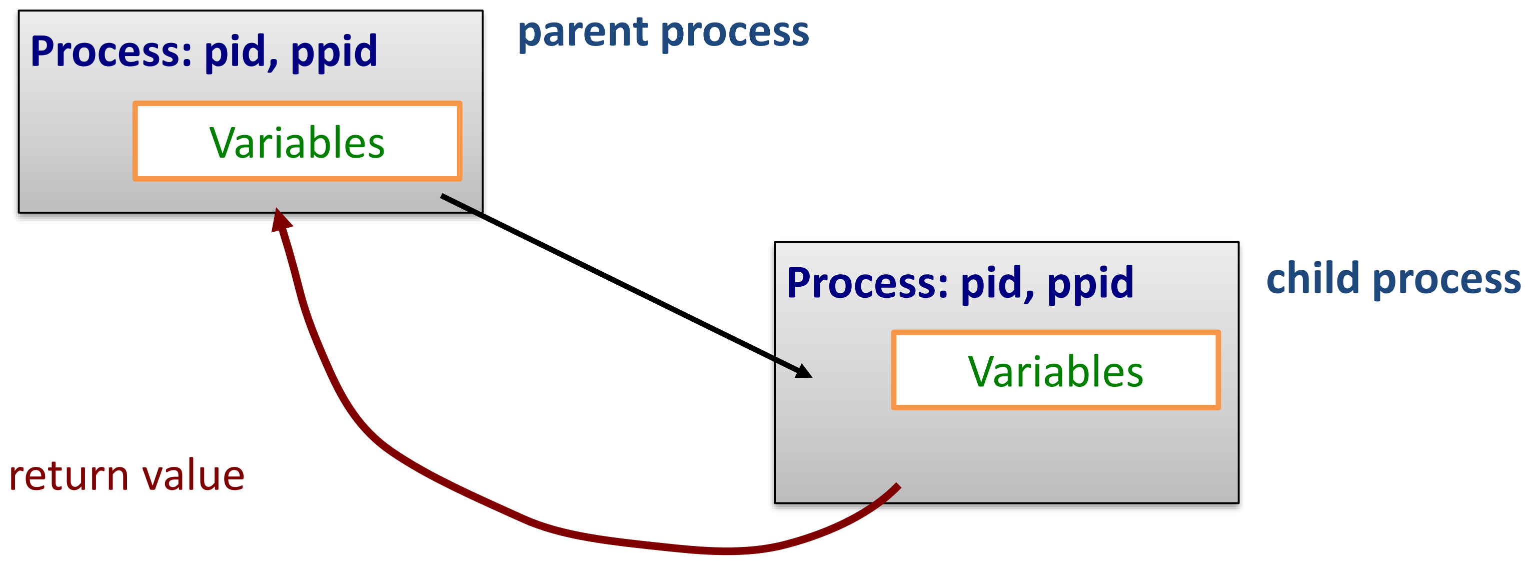
Compact notation:

```
if  command1;  then
        command2
        ...
fi
```

```
if  command1;  then
        command2
        ...
else
        command3
        ...
fi
```

Process Return Value

Exiting process can share the information about its execution to its parent process by passing back **return value**. The return value is an integer in the range 0-255.



Return value:

0 = everything was successful (true)

> 0 = error has occurred

the return value then generally identifies the error **(false)**

The return value of the last executed command is stored in **?** variable.

Return Value, Examples

```
$ mkdir test
$ echo $?
0
```

```
$ mkdir test
mkdir: cannot create directory `test': File exists
$ echo $?
1
```

```
$ expr 4 + 1
5
$ echo $?
0
```

```
$ expr a + 1
expr: non-integer argument
$ echo $?
1
```

test, Operators for Comparing

The **test** command is used to compare values and to test types of files and directories (man bash, man test). If the test passes successfully, the return value of the command is set to 0 (true).

Comparison of Integers:

```
test number1 operator number2
```

Operator :

```
-eq    equal to
-ne    not equal to
-lt    less than
-le    less than or equal to
-gt    greater than
-ge    greater than or equal to
```

Additional information:
man bash, man test

Alternative notation:

must be spaces

```
[[ number1 operator number2 ]]
```

test, Logical Operators

Logical operators:

```
||     logical or
&&     logical and
!      negation
```

same result, but different way of interpretation!

Příklady:

```
[[ (number1 operator number2) || (number3 operator number4) ]]
```

~~`[[ (number1 operator number2) ]] || [[ (number3 operator number4) ]]`~~

We do not recommend

- Logical operators can be used to create more complex conditions.
- If we do not know priority of the operators, we should use parentheses.
- Bash **uses** lazy evaluation of conditions, which is based on evaluating only the component of the logical condition that must be evaluated to determine the logical value of the whole condition.

Lazy Evaluation

```
[[ expression1 || expression2 ]] <-> [[ expression1 ]] || [[ expression2 ]]
```

```
F || F = F
F || T = T
T || F = T
T || T = T
```

If the expression1 is true (**T**), the overall result is always true. Therefore expression2 is evaluated only if first expression is not true.

Trick:

```
mkdir directory || exit 1
```

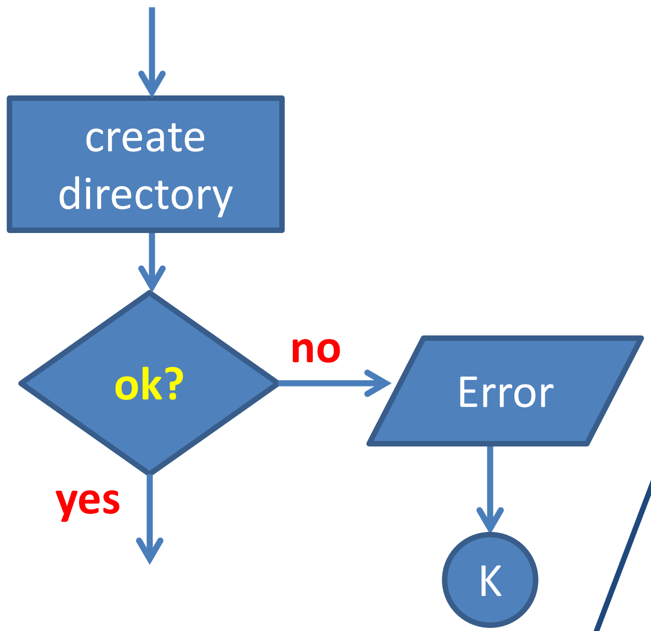
If command mkdir fails (F), calls the exit and terminates script

```
[[ expression1 && expression2 ]] <-> [[ expression1 ]] && [[ expression2 ]]
```

```
F && F = F
F && T = F
T && F = F
T && T = T
```

If the expression1 is false (**F**), the overall result is always false. Therefore expression2 is evaluated only if first expression is true

Practical Example - Condition



Functionally identical notation

```
mkdir dir 2> /dev/null
if [[ $? -ne 0 ]]; then
     echo "Can't  create directory!"
     exit 1
fi
```

```
mkdir dir 2> /dev/null
if test $? -ne 0; then
     echo "Can't  create directory!"
     exit 1
fi
```

```
if ! mkdir dir 2> /dev/null; then
     echo "Can't  create directory!"
     exit 1
fi
```

Cycle via while/until

Cycle (loop) is a control structure that repeatedly executes a series of commands. Both repeating and exiting of the cycle is managed by a condition.

```
while  command1
       do
              command2
              ...
       done
```

cycle runs **while** return value of command1 is **0 in return** (no error)

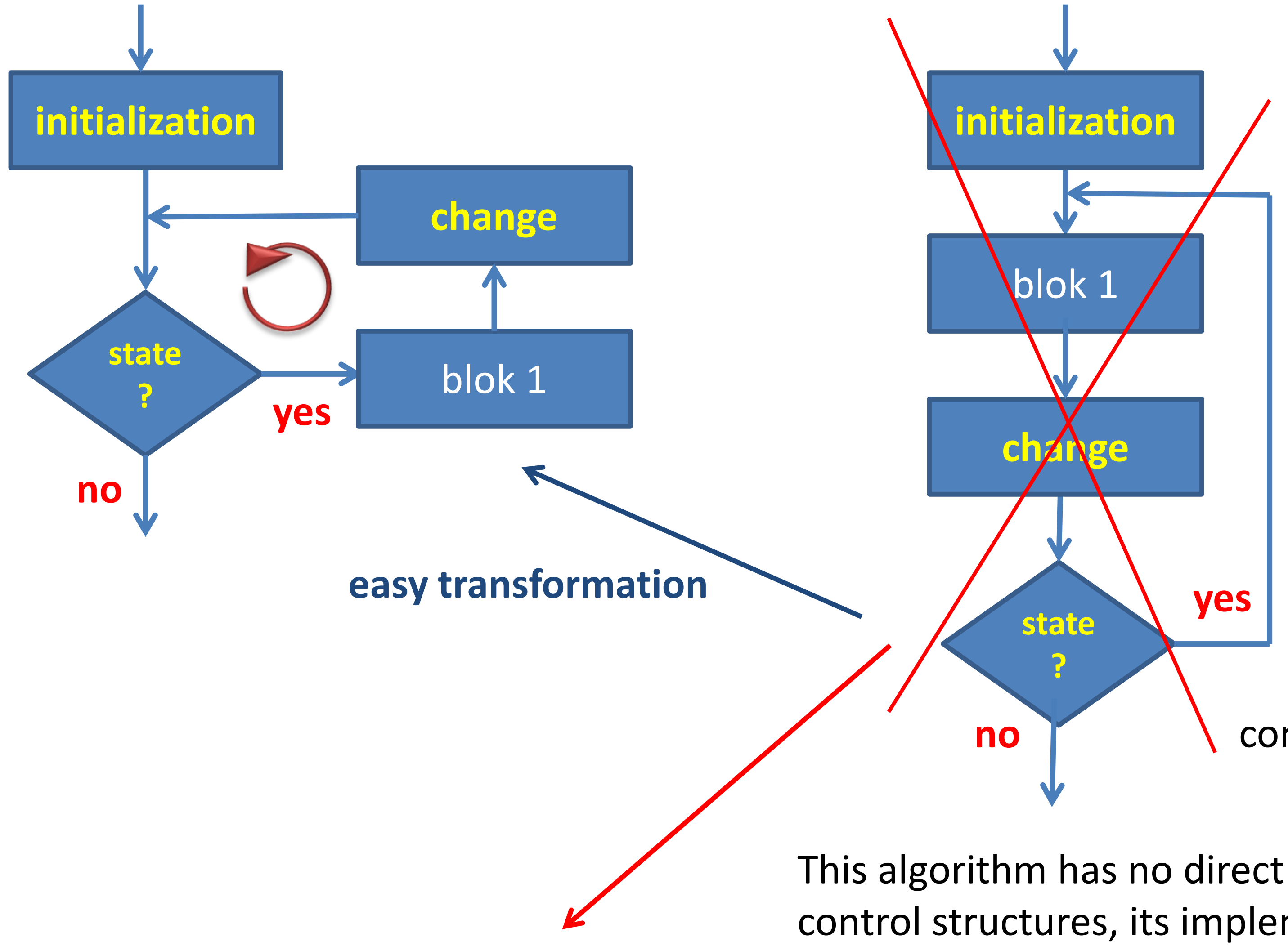
cycle runs **until return value** of command1 is **0**

```
until  command1; do
       command2
       ...
done
```

Compact notation:

```
while  command1; do
       command2
       ...
done
```

Cycle via while/until...



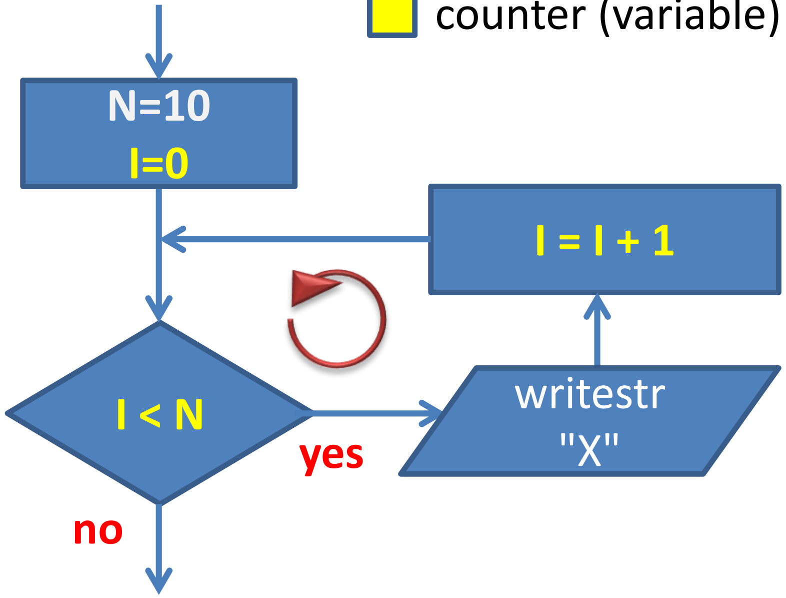
easy transformation

Evaluation of the condition at the end of cycle

complicated implementation

This algorithm has no direct support in bash control structures, its implementation is possible, but at the cost of worse readability of the final code.

Practical Example - Cycle



$ must be uses

```
N=10
I=0
while test "$I" -lt "$N"; do
        echo "X"
        ((I = I + 1))
done
```

```
N=10
I=0
while [[ I -lt N ]]; do
        echo "X"
        ((I = I + 1))
done
```

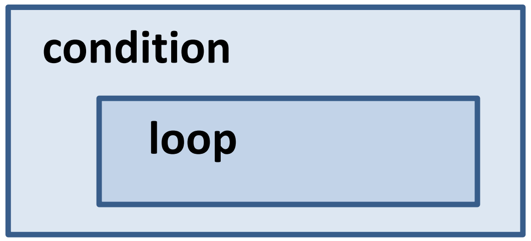
```
N=10
I=0
while [[ "$I" -lt "$N" ]]; do
        echo "X"
        ((I = I + 1))
done
```

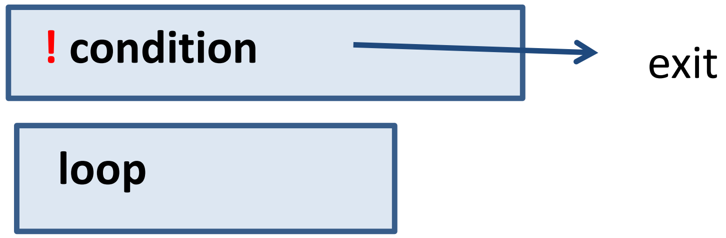
$ is optional when using [[ ]] or (( )) block

More Complex Structures - Nesting

Bash does not embody **labels** and **goto command**, nor its equivalent. Thus, it is necessary to use nested loops or conditions to create more complex structures. The level of nesting is not limited.

We try to avoid unnecessary nesting in script/algorithm design (mostly for easier orientation in the script).





Better arrangement of blocks, e.g. for testing user data input.

exit Command

exit command is used to terminate the execution of the script or the interactive session. Optional argument of the command is **return value**.

```
#!/bin/bash
if test "$1" -lt 0; then
     echo "The number is less than zero!"
     exit 1
fi
echo "The number is greater than or equal to zero."
exit 0
```

```
$ ./my_script 5
The number is greater than or equal to zero.
$ echo $?
0
```

```
$ ./my_script -10
The number is less than zero!
$ echo $?
1
```

Nesting Cycles - Example

```
N=10
I=0
while [[ I -lt N ]]; do
     J=0
     while [[ J -lt I ]]; do
          echo -n "X"
          ((J = J + 1))
     done
     echo ""
     ((I = I + 1))
done
```

counter of outer cycle can affect behavior of the inner cycle

Indentation of text blocks leads to increased clarity and **readability** of the code and should be maintained especially with nested structures.
Indentation is usually supported in text editors. For example, **gedit** can offset the marked text block by shortcut TAB or Shift + TAB.

Exercise I

1. Write bash scripts for the following tasks. Size of the plotted shape is entered interactively by the user after launching of the script.

Task 1

Print a square composed from **X** characters to the terminal. Side length of the square is entered by the user.

```
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
```

Please ignore the fact, that it is not visually a square. Number of **X** characters per line and the number of lines must be the same.

Task 2

Print a triangle composed from **X** characters to the terminal. Legs of triangle are placed at left and top of the triangle. Leg length of the triangle is entered by the user.

```
X X X X X X X X X X
X X X X X X X X X
X X X X X X X X
X X X X X X X
X X X X X X
X X X X X
X X X X
X X X
X X
X
```

Task 3

Print a triangle composed from **X** characters to the terminal. Legs of triangle are placed at left and bottom of the triangle. Leg length of the triangle is entered by the user.

```
X
X X
X X X
X X X X
X X X X X
X X X X X X
X X X X X X X
X X X X X X X X
X X X X X X X X X
X X X X X X X X X X
```

Task 4

Print a square outline composed from **X** characters to the terminal. Side length of the square is entered by the user.

```
X X X X X X X X X X
X                 X
X                 X
X                 X
X                 X
X                 X
X                 X
X                 X
X                 X
X X X X X X X X X X
```

Please ignore the fact, that it is not visually a square. Number of **X** characters per line and the number of lines must be the same.

Task 5

Print a square outline and its diagonals composed from **X** characters to the terminal. Side length of the square is entered by he user.

```
X X X X X X X X X X
X X             X X
X   X         X   X
X     X     X     X
X       X X       X
X       X X       X
X     X     X     X
X   X         X   X
X X             X X
X X X X X X X X X X
```

Please ignore the fact, that it is not visually a square. Number of **X** characters per line and the number of lines must be the same.

Homeworks

Homeworks

Instructions:

1. Listed tasks are **for advanced students**.
2. **The goal of the tasks is to develop your ability to solve problems that are seemingly unsolvable from the point of available options and resources.** In case of bash language, this involves mainly the possibility to work only with integer arithmetic and limited way of rendering into the terminal

Tasks:

1. Draw a circle using character "X". The radius of the circle is entered by the user after starting of the script.
2. Draw a circle outline using character "X". The radius of the circle is entered by the user after starting of the script