

C2110 *UNIX and Programming*

9th lesson

Bash - Completion (Almost)

Petr Kulhánek

kulhanek@chemi.muni.cz

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

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

Contents

➤ **Command test**

- **comparing integers and strings**

➤ **Loops**

- **for vs while, for in, pipes and redirection**

Command test, integers

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).

Integer comparison:

```
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

| | |
|------------------|-------------------------------------|
| != | not equal to |
| == | equal to |
| < | less |
| <= | less then or equal to |
| > | greater then |
| >= | greater then or equal to |

Additional information:
man bash, man test

Alternative notation:

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

must be spaces



Command test, strings

Comparison of strings

```
test string1 operator string2  
[[ string1 operator string2 ]]
```

Operator :

== strings are identical (= also can be used)
!= strings are different

Testing strings

```
test operator string1  
[[ operator string1 ]]
```

Operator :

-n tests whether the string **has not** zero length
-z tests whether the string **has** zero length
-f tests whether the string has name of an existing **file**
-d tests whether the string has name of an existing **directory**

Command test, logical operators

Logical operators:

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

- 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.

Command test, examples

```
[[ (I -ge 5) && (I -le 10) ]]
```

Is the value of the variable I in the range <5;10>?

```
[[ (I -lt 5) || (I -gt 10) ]] or [[ !((I -ge 5) && (I -le 10)) ]]
```

Is the value of the variable I out of the range <5;10>?

```
[[ I -ne 0 ]]
```

Is the value of the variable I different than zero?

```
[[ "$A" == "test" ]]
```

Does the variable contain string "test"?

```
[[ "$A" != "test" ]]
```

Does the variable contain different string than string "test"?

```
[[ -z "$A" ]]
```

Does the variable contain an empty string?

```
[[ -f "$NAME" ]]
```

Is there a file with same name as value of NAME?

```
[[ ! (-d "$NAME") ]]
```

Is there no directory with name as the value of NAME?

Exercise I

1. Write a script that will ask the user to gradually enter two numbers. Then, it will print quotient of the two numbers. Treat a possible error of division by zero.
2. Write a script that will create a directory, name of directory is entered by the user after starting the script. Treat the error situation caused by already existing directory.
3. Write a script that asks you to enter an integer. The script will then test whether this is indeed an integer.

Loops: FOR

Loop is a control structure that repeatedly performs a series of commands. Both repeating and exit from the loop is managed by condition.

It is executed before starting the loop (counter initialization)

if the condition is true, command1 and others commands are executed.

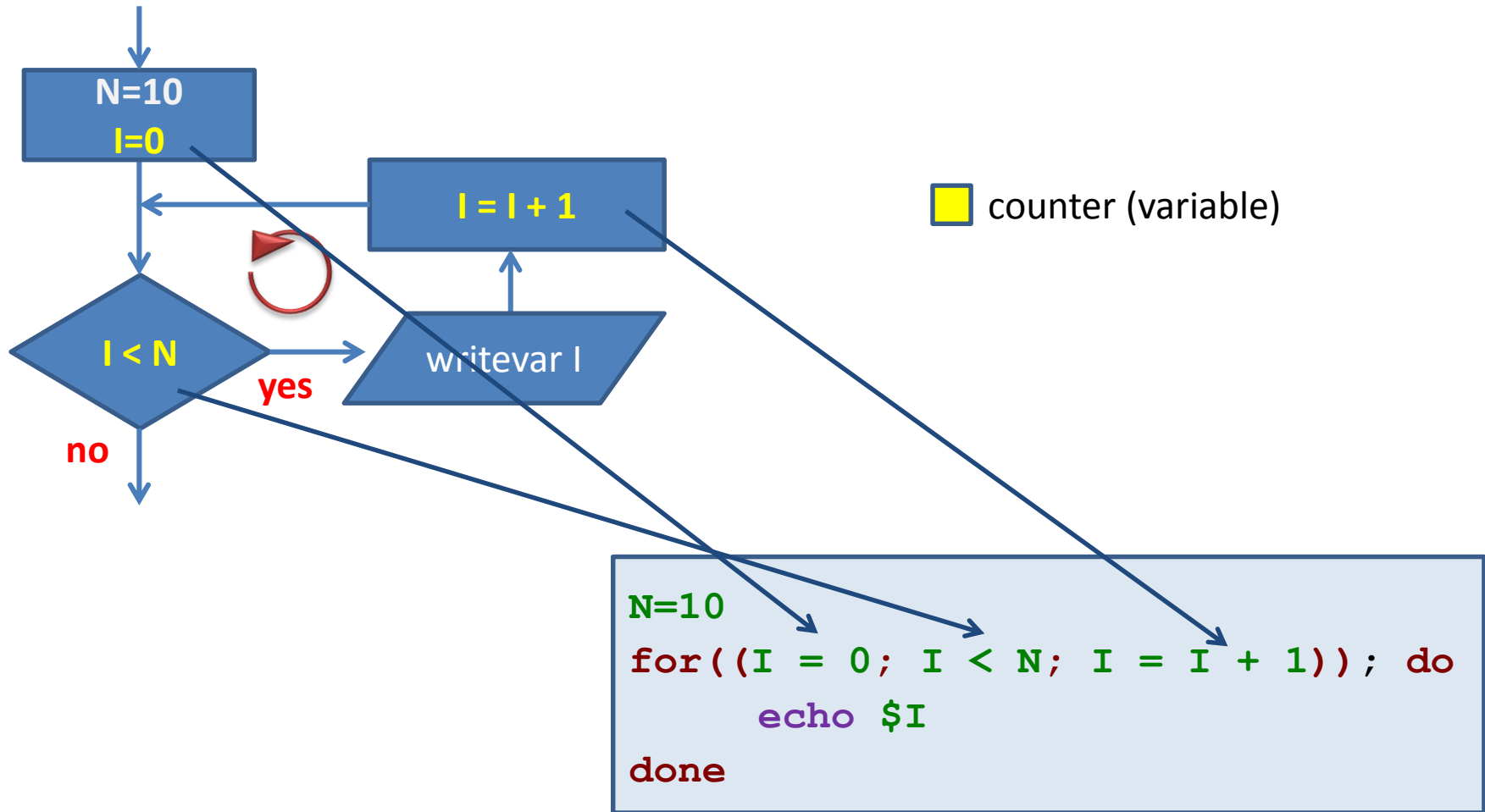
```
for( (initialization; condition; change) )
do
    command1
    ...
done
```

Compact notation:

```
for( (initialization; condition; change) ); do
    command1
    ...
done
```

The counter is updated after the execution of commands.

FOR loop and flowchart



Loop: FOR vs WHILE

```
for ((I=1; $I <= 10; I++)); do  
    echo $I  
done
```

performed before start of the loop
(counter initialization)

In addition to change of the counter at the end of loop, it is possible to execute other changes in the body of the loop. This is **not recommended** because it reduces code readability.

if the condition is true, executes commands in the do/done block

```
I=1  
while [[ $I -le 10 ]]; do  
    echo $I  
    (( I = I + 1 ))  
done
```

Update counter after the execution of commands

Counter update can be done **anywhere** in the body of loop (even at multiple locations).

FOR loop usage

Prints numbers from 1 to 10

```
for((I=1;I <= 10;I++)); do
    echo $I
done
```

Variable **I** is **counter**.

Initialization is governed by rather free rules because the expression is in (()) block.

Change:

Possible interpretation of the all mathematical expression that can be interpreted within (()) block, e.g.:

- ++ value is increased by one
- value is decreased by one
- others ...

Prints numbers from 10 to 1

```
for((I=10;I >= 1;I--)); do
    echo $I
done
```

Condition:

Following comparison operators can be used:

| | |
|----|--------------------------|
| != | not equal to |
| == | equal to |
| < | less |
| <= | less then or equal to |
| > | greater then |
| >= | greater then or equal to |

can be used only with integers in (())

FOR loop, change of counter

If the variable can be interpreted as integer, following arithmetic operators can be used:

++ variable value incremented by one

A++

-- variable value decremented by one

A--

+ sums up two values

A = 5 + 6

A = A + 1

- subtracts two values

A = 5 - 6

A = A - 1

***** multiplies two values

A = 5 * 6

A = A * 1

/ divides two values (integer division)

A = 5 / 6

A = A / 1

A=A+3

+= increases variable by value

A += 3

A += B

-= decreases variable by value

A -= 3

A -= B

***=** multiplies variable by value

A *= 3

A *= B

/= divides variable by value

A /= 3

A /= B



Nested loops

Loops can be freely nested.

```
for((I=1;I <= 10;I++)); do
  for((J=1;J <= 10;J++)); do
    echo "$I $J"
  done
done
```

Outer loop

Inner loop

```
for((I=1;I <= 10;I++)); do
  for((J=1;J <= I;J++)); do
    echo "$I $J"
  done
done
```

counter of outer loop can affect behavior of inner loop

The level of nesting is not limited. For loop can be combined with other loops (while, until, for in) or conditions.

Exercise II

1. Write bash scripts for Task 1 and 2. Use for loops instead of while loops. Size of plotted shape is given as the first argument of the script. The script will tests if the entered number of arguments is correct and if the first argument is an integer greater than zero.
2. Edit the task 1, that the rectangle is printed into the terminal. The size should be entered interactively.

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
X X
X
```

Loop FOR ... IN ...

Commands in the block **do/done** (command1, ...) are executed for each element in the list **LIST**. In the given run of the loop, variable **VAR** includes current element from the list **LIST**

```
for VAR in LIST
do
    command1 $VAR
    ...
done
```

Compact notation:

```
for VAR in LIST; do
    command1 $VAR
    ...
done
```

Loop FOR ... IN ..., lists

```
for A in a b c; do
    echo $A
done
```

Loops will run three times, characters **a**, **b**, **c** are printed, one character in each run of cycle.

It is appropriate to create the list of items algorithmically (using the commands listed in graves - reverse apostrophes).

```
for A in `ls *.eps`; do
    ./process_file $A
done
```

Command **process_file** is executed for every file with suffix **eps**, which is in the current directory.

```
for A in `seq 1 0.25 10`; do
    printf "%8.3f\n" $A
done
```


Creates real numbers in the range of 1 to 10 with an increment of 0.25. The numbers will be printed with an accuracy of three decimal places aligned right with 8 character length.

Documentation: man seq

Redirection and pipes

Read file line by line:

```
cat file.txt | while read A; do
    command2
    ...
done
```



pipe


```
while read A; do
    command2
    ...
done < file.txt
```



redirection

Redirection to file:

```
for((I=1;I <= 10;I++)); do
    echo $I
done > file.txt
```



Output of all commands in the loop is redirected to **file.txt**.

Redirection and pipes - examples

```
for((I=1;I <= 10;I++)); do  
    echo $I  
done > file.txt
```

Same functionality

```
rm -f file.txt  
for((I=1;I <= 10;I++)); do  
    echo $I >> file.txt  
done
```

```
for((I=1;I <= 10;I++)); do  
    echo $I  
    printf "N=%10d\n" $I  
done > file.txt
```

Different functionality

```
rm -f file.txt  
for((I=1;I <= 10;I++)); do  
    echo $I >> file.txt  
    printf "N=%10d\n" $I  
done
```

Exercise III

1. Edit the scripts from the previous exercise in such a way that size of the shapes would be read from standard input and the resulting shape will be printed to a file, the name of this file is entered to standard input by user
2. Write a script that prints real numbers in the range from -10 to 10 with an increment of 0.5. Numbers will be printed including the sign, right-aligned, values will be 10 characters long and with precision one decimal point.


Homeworks



Homework I


Explain different behavior of the following scripts. Data.txt file contains five lines.

```
#!/bin/bash
I=0
cat data.txt | while read A; do
    I=$((I+1))
done
echo $I
```



prints number 0

```
#!/bin/bash
I=0
while read A; do
    I=$((I+1))
done < data.txt
echo $I
```



print number 5

Homework II

File `rst.out` (`wolf.ncbr.muni.cz:/home/kulhanek/Documents/C2110/Lesson09/rst.out`) contains the results of molecular dynamics simulation. Your task is to extract the temperature dependence of the simulated system on time from the file, you will save these data to file `temp.out`, which will contain two columns. First column will be time and second column will be temperature.

```
.....
NSTEP =      500    TIME (PS) =      0.500    TEMP (K) =    288.02    PRESS =      0.0
Etot   =      942.6248    EKtot   =      151.0990    EPtrot   =      791.5258
BOND   =      51.3204    ANGLE  =      292.3619    DIHED    =      176.5980
1-4 NB =      17.7099    1-4 EEL =      981.4071    VDWAALS  =      -68.3301
EELEC  =     -494.7423    EGB    =     -164.7991    RESTRAINT =      0.1822
EAMBER (non-restraint) =      791.3436
.....
```

time temperature

Caution: It is forbidden to use commands `grep`, `awk`, and also their variants in the script. To solve the task use commands `read` and `while`.

Selfstudy

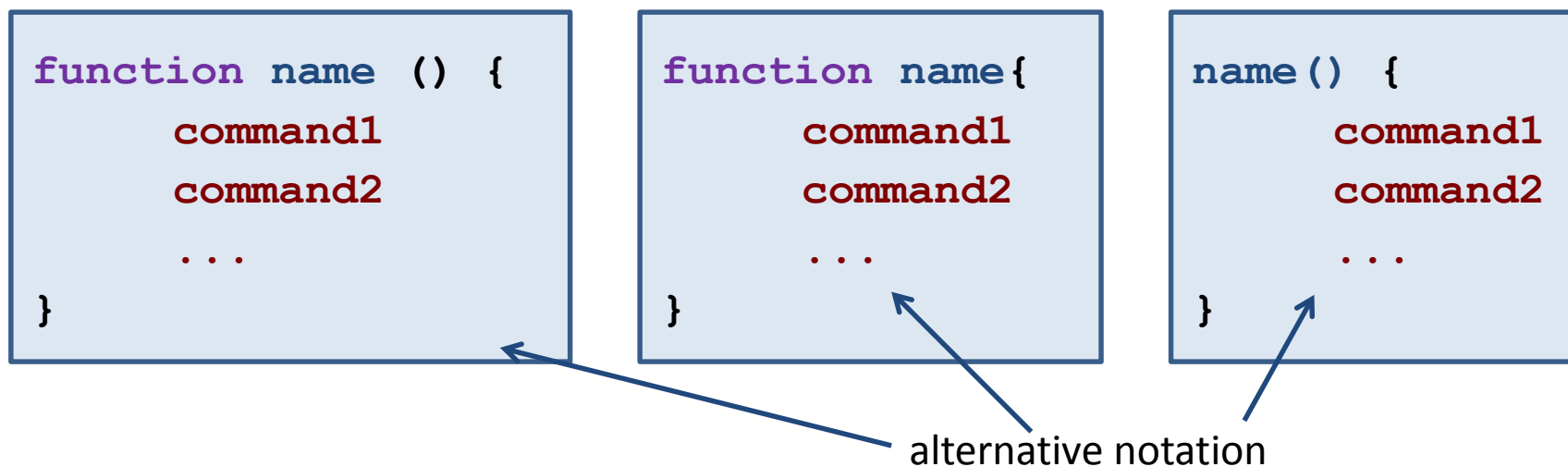
functions - **for advanced students**



Functions - definition

Function is a structure that allows to group part of the code so that it can be easily used on multiple locations of the script. Functions result in clean and readable code when some tasks are repeated.

Definition:



Arguments of the function are not declared, therefore there is no control of the number of the given arguments, no type control, and functions can not be overloaded. Given arguments are available via special variables `#`, `1-9`, `*`. The functions are executed as an existing command. **Variables in the function are global** (it can be changed by using the keyword `local`).

Documentation: `man bash`, sekce `FUNCTIONS`.

Function - use

```
# print line - the length is in the first argument
function print_line () {
    N=$1
    for((J=1;J <= N;J++)); do
        echo -n " X"
    done
    echo ""
}

# use function
print_line 10 # print line 10 characters long
print_line 5  # print line 5 characters long
```

Value of the argument is available via special variable **1**

Exercise

1. Write a script that will print the square, and triangle (similar to task 1 and 2) for one specified length one after another to the terminal. Identify the part that is repeated and rewrite it using function.

```
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.