C2110 UNIX and programming

12th Lesson

Petr Kulhánek, Jakub Štěpán

kulhanek@chemi.muni.cz

National Centre for Biomolecular Research, Faculty of Science
Masaryk University, Kotlářská 2, CZ-61137 Brno

european social fund in the czech republic

EUROPEAN UNION

MINISTRY OF EDUCATION, YOUTH AND SPORTS

OP Education for Competitivness

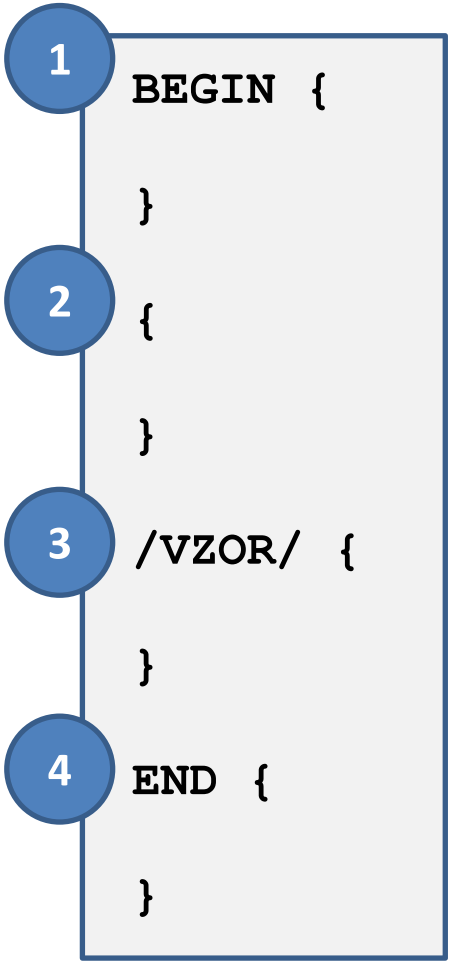
INVESTMENTS IN EDUCATION DEVELOPMENT

CZ.1.07/2.2.00/15.0233

Contents

- **AWK**
  - **Text files analysis**
  - **Regular expressions**
  - **Arrays**
- **BASH user input check**
  - **BASH error states**
  - **Input values check**

Script execution



```
BEGIN {

}
{

}
/VZOR/ {

}
END {

}
```

- Block BEGIN (1) is executed (if present) before file analysis.
    - **Record** from file is read. By default one record is whole line from input file or stream. Record is split to **fields**. By default words of line are fields.
    - Block (2) is executed for **any record**.
    - Block (3) is executed for any **record matching PATTERN**.
    - .... Possible other blocks are executed ....
- Block END (4) is executed (if present) after analyzing whole file content.

Each block is in curly brackets {}.
Some program blocks are optional – see description.
Default record separator is new line – one line = one record.

Regular expressions

```
/PATTERN/ {

}
```

If record matches PATTERN, then block is executed.

PATTERN may be **regular expression**.

Regular expression is string construction, that describes structure of set of text strings. It may be used to search text strings or substitutions of substrings.

Simple regular expressions samples:

TEXT - matches if record contain text TEXT (TEXT may occur **anywhere** in record)
^TEXT - matches if record contain text TEXT on record beginning
TEXT$ - matches if record contain text TEXT on record end

Exercise

1. Extract temperature dependency on time from file **rst.out**. Display graph of dependency by **gnuplot**.

```
NSTEP =     1000   TIME(PS) =       1.000  TEMP(K) =   305.69  PRESS =     0.0
 Etot   =       907.8481  EKtot   =       160.3711  EPtot      =       747.4770
 BOND   =        40.6154  ANGLE   =       273.9238  DIHED      =       164.5827
 1-4 NB =        14.6900  1-4 EEL =       973.2602  VDWAALS    =       -67.6091
 EELEC  =      -488.9232  EGB     =      -163.0629  RESTRAINT  =         0.3793
 EAMBER (non-restraint)  =       747.0977
```

2. Extract time dependency of energies from file **rst.out**. Extract total energy (**Etot**), kinetic energy (**EKtot**) and potential energy (**EPtot**) time dependency. Display graphs of all energies in **gnuplot**. Make sure, that sum of potential and kinetic energy is equal to total energy.

Arrays

AWK provides associative arrays. An array has name, all items are accessed by key. Key may have arbitrary type and value. Key may be variable value.

Value assignment:

```
        my_array[key] = value;
```

Obtain value:

```
        variable = my_array[key];
```

Examples:

```
        i = 5;
        my_array[i] = 15;
        print my_array[i];

        a = "word";
        my_array[a] = "value";
        print my_array["word"], my_array[5];
```

Arrays, ...

Searching in key list:

```
for( variable in array) {
      print array[variable];
      ...
}
```

Cycle does one iteration for each key value used in **array**. Actual key value is in **variable**.

Array item deletion by key:

```
delete array[key];
```

Exercise

1. Extract **temperature time dependency** from file **rst.out**. Remove last 2 values (these are average and fluctuation). Display **graph in gnuplot**.

2. Extract **temperature** values from file **rst.out** and calculate its **average value**. Compare calculated value with value printed in file rst.out. **Why both values differ?**

BASH user input check

- BASH error states
- Input values check

BASH error states

Example from lession 8. Script does **not behave correctly** if started with **no argument** or **with non-numerical argument**.

```
#!/bin/bash
if test "$1" –le 0; then
     echo "Number not greater then zero!"
     exit 1
fi
echo "Number greater then zero."
exit 0
```

```
$ ./my_script
my_script: line 2: test: -le: unary operator expected
Number greater then zero.
$ echo $?
0
```

```
$ ./my_script f
my_script: line 2: test: f: integer expression expected
Number greater then zero."
$ echo $?
0
```

Input values check

It is **neccessary to check** values obtained from **user**.

Check

- Number and type of arguments
- Validity of numerical values (zero devision, negative counter values)
- Zero string length
- Existence of files for processing

```
#!/bin/bash
echo "Write numeric value!"
read A
if ! expr $A + 0 > /dev/null; then
          echo „Error! Non-numeric value read: $A"
          exit 1
fi
```

Exercise

1. To script from home work 1 in lesson 8 (rectangle drawing) **add check** that user submitted **exactly two** arguments.
2. Adjust previous script to check that user submitted size in **natural numbers**.
3. Adjust previous script in such a way, that **third argument** will be character or string to print rectangle with (instead of character "X"). Check if argument is non-empty string.
4. Adjust previous script in such a way, that user will insert values **interactively** on request **after script start**.
5. Adjust script from home work II lesson 9 in such a way, that script will accept **name of analyzed file as a argument** and file **path existence** will be checked.