

OPERATING SYSTEMS

– Laboratory 4 –

1. REGULAR EXPRESSIONS

- a **regular expression** (*regexp*) = a finite character sequence defining a search pattern
- a **match** = a single character, a sequence of characters, a sequence of bytes, a piece of text
- the special characters (meta-characters):

.	period (dot)	\	backslash	^	caret	\$	dollar sign
	vertical bar	?	question mark	*	asterix (star)	+	plus sign
(	opening parenthesis	)	closing parenthesis	[	square bracket	{	curly brace

- these special characters have a different meaning in regular expressions
 - you need to escape them (using \ backslash) in order to restore their own regular meaning or to give them the special meaning (see example like \?)
- the meaning of special characters in regular expressions:

Expression	Matches
.	any single char
\.	the . (dot) char
[abc]	a single char inside square brackets (a, b or c)
[^abc]	a single char EXCEPT those inside square brackets (d, e, ..., z)
[a-z]	a single lowercase letter from a to z (any lowercase letter)
[A-Z]	a single uppercase letter from A to Z (any uppercase letter)
[a-zA-Z]	a single lowercase or uppercase letter
[0-9]	a single digit from 0 to 9
[^0-9]	a single char which IS NOT digit
\d	a single digit from 0 to 9 (equivalent with [0-9])
\s	a single whitespace char (including SPACE, TAB, CR, LF)
\w	a single alphanumeric char or _ (underscore)
\(\)	capture a group between parenthesis

- example:

1. Given the following text lines:

```
abc
bdf
ceg
```

you can write some regular expressions to match:

- only the first line: 'abc'
- only the second line: 'bdf'
- only the third line: 'ceg'
- all the lines above: '...' or, much better, '[abc][bde][cfg]'

- anchors:

Symbol	Matches
<code>^</code>	the start of line
<code>\$</code>	the end of line
<code>\<</code>	the empty string at the beginning of a word
<code>\></code>	the empty string at the end of a word
<code>\b \b</code>	equivalent with <code>\< \></code>

- repetition operators:

Operator	Meaning
<code>\?</code>	either zero or one time
<code>*</code>	zero or more times
<code>\+</code>	one or more times
<code>\{n\}</code>	exactly <i>n</i> times
<code>\{n, \}</code>	<i>n</i> times or more
<code>\{, m\}</code>	at most <i>m</i> times
<code>\{n, m\}</code>	at least <i>n</i> times, but at most <i>m</i> times

- example:

2. Given the following text lines:

```
aaabc
aaadf
aaace
```

you can write a regular expression to match:

- all the lines above: `'aaa[bdc][cfe]'` or `'a\{3\}[bdc][cfe]'`

2. grep

- searches the input file and prints all the lines which contain the given pattern
- its name is derived from "global regular expression print"
- command syntax:

```
grep [OPTIONS] PATTERN [FILE...]
```

```
grep [OPTIONS] [-e PATTERN] [-f FILE...] [FILE...]
```

- OPTIONS:

<code>-c (--count)</code>	print a count of matching lines
<code>-i (--ignore-case)</code>	ignore case distinctions
<code>-v (--invert-match)</code>	invert the sense of matching
<code>-A NUM (--after-context=NUM)</code>	print NUM lines after matching lines
<code>-B NUM (--before-context=NUM)</code>	print NUM lines before matching lines
<code>-C NUM (-NUM --context=NUM)</code>	print NUM lines from all matching lines

- PATTERN* is usually provided in the command line using a regular expression
- to specify multiple search patterns, or to protect a pattern beginning with a *hyphen* (`-`):

-e *PATTERN* (--regexp=*PATTERN*)

- to obtain patterns from *FILE* (one pattern per line):

-f *FILE* (--file=*FILE*)

3. sed (Stream Editor)

- is a non-interactive text editor used to perform basic text transformations on an input stream
- reads and process all lines of the input stream one by one, and prints the result on the screen
- command syntax:

```
sed [-n] [-e] ' [/pattern/]command' [input-file]
```

```
sed [-n] -f script-file [input-file]
```

-n suppress automatic printing of internal buffer (*pattern space*)

-e *script* add *script* to the commands to be executed

-f *script-file* add the contents of *script-file* to the commands to be executed

– the input stream may be: the standard input stream (keyboard), a file denoted by *input-file* or the result of another command(s) execution

– if not specified a pattern, a certain line, or multiple lines, *command* will be executed on all the lines of input stream

- selecting lines (line addressing):

N just line N

\$ just last line

M, N from line M to line N

M~step from line M, lines from step to step

/regexp/ just the lines containing the pattern given by *regexp*

0, /regexp/ just the first line containing the pattern given by *regexp*

M, +N from line M, N lines after

M, ~N from line M, all the lines which are multiple of N

- commands:

– **p** (print)

```
sed angajati.txt
```

```
sed 'p' angajati.txt
```

```
sed -n 'p' angajati.txt
```

```
sed -n '2p' angajati.txt
```

```
sed -n '/Tudor/p' angajati.txt
```

```
sed -n '2,5p' angajati.txt
```

```
sed -n '/Ion/,/Victor/p' angajati.txt
```

```
sed -e '2p' -e '5p' angajati.txt
```

– **d** (delete)

```
sed 'd' angajati.txt
```

```
sed '4d' angajati.txt
```

```
sed '/Tudor/d' angajati.txt
sed '2,5d' angajati.txt
sed '/Tudor/, $d' angajati.txt
sed -e '2d' -e '5d' angajati.txt
```

– **s (substitute)** `s/regex/repl/[gi]` : substitute first occurrence of regex in a line with repl;
flags g for global replacement (all matching in the line), **i** for case insensitive regex

```
sed 's/Tudor/Tudorel/' angajati.txt
sed -n 's/Tudor/Tudorel/' angajati.txt
sed -n 's/19/18/g' angajati.txt
sed -n 's/1931/1932/p' angajati.txt
sed -n 's/\(Ion\)el/\1ut/p' angajati.txt
sed -n 's/[0-9]\|[0-9]\$/&\.5/' angajati.txt
sed -n '/Olga/,/Toma/s/$/**CONCEDIU**/' angajati.txt
```

– **a (append)**

```
sed '3a Linie adaugata' angajati.txt
sed '$a TERMINAT' angajati.txt
sed '/Adrian/a Linie adaugata' angajati.txt
```

– **c (change)**

```
sed '2c SALARIAT PENSIONAT' angajati.txt
```

– **i (insert)**

```
sed '1i \t\t\tDATE DESPRE PERSONAL' angajati.txt
```

– **q (quit)**

```
sed '5q' angajati.txt
```

– **r (read content from file)**

```
sed '3r text.txt' angajati.txt
```

– **w (write content to file)**

```
sed -n 'w angajati.bak' angajati.txt
```

– **= (print line number)**

– **l (display control characters)**

```
sed -n 'l' test.txt
```

– **n (next)**

– **y (transform)** - ex. replace each lowercase vowel with the corresponding uppercase vowel
(the initial list and replacement list must have the same length, performs 1 to 1 transliteration)

```
sed 'y/aeiou/AEIOU/' text.txt
```

– **h (holding)**

– **g (getting)**

– **x (exchange)**

4. awk

- is not only a text processing utility, but also an interpreted programming language with a C-like syntax
- its name is derived from its creators: Alfred Aho, Peter Weinberger, Brian Kernighan
- command syntax:

```
awk [OPTIONS] '/pattern/' [input-file]
```

```
awk [OPTIONS] '{action}' [input-file]
```

```
awk [OPTIONS] '/pattern/{action}' [input-file]
```

`-F fs` to change the default input field separator with `fs`

`-f script-file` to obtain the commands from `script-file`

- awk reads and process all lines of the input file one by one
- each line represents an input record
- default input record separator: CR (Carriage Return)
- the current input record is stored in the internal variable `$0`
- each input record is parsed and separated into chunks called fields
- default input field separators: SPACE or TAB
- built-in variables:

`$0` the current input record

`$1, $2, ...` the fields of the current input record

`NR` the total number of input records seen so far

`NF` the number of fields in the current input record

`RS` the input record separator

`ORS` the output record separator

`FS` the input field separator

`OFS` the output field separator

`OFMT` the format for converting numbers to strings for printing with `print`

`ARGC` the number of command line arguments

`ARGV` the array of command line arguments

`FILENAME` the name of the current input file

`FNR` the current record number in the current file

`ENVIRON` the array of environment variables

- examples:

- print all lines of the input file:

```
awk '{print}' angajati.txt
```

```
awk '{print $0}' angajati.txt
```

- print all lines which contain the given pattern:

```
awk '/Tudor/' angajati.txt
```

```
awk '/Tudor/{print}' angajati.txt
```

```
awk '/Tudor/{print $0}' angajati.txt
```

- change the default input field separator:

```
awk -F: '{print $1}' /etc/passwd
```

```
awk -F: '{print NR, $1}' /etc/passwd
```

```
awk -F'[ :\t]' '{print $1, $2, $3}' angajati.txt
```

- **relational operators:**

Operator	Name	Example
<	less than	$x < y$
<=	less than or equal to	$x \leq y$
==	equal	$x == y$
!=	not equal	$x != y$
>	greather than	$x > y$
>=	greather than or equal to	$x \geq y$
~	matches the regular expression	$x \sim /regex/$
!~	does not match the regular expression	$x !\sim /regex/$

- **examples:**

- using relational operators:

```
awk '$5 < 2000' angajati.txt
```

```
awk '$5 < 2000 {print}' angajati.txt
```

```
awk '$5 == 1942 {print NR, $1}' angajati.txt
```

- using relational operators and regular expressions:

```
awk '$1 ~ /Tudor/ {print}' angajati.txt
```

```
awk '$1 !~ /Tudor/ {print}' angajati.txt
```

- **logical operators:** && || !

- **arithmetic operators:** + - * / % ^

- **assignment operators:** = += -= *= /= %= ^=

- **conditional expressions:**

```
condition ? expresion1 : expresion2
```

is equivalent with:

```
if (condition)
```

```
    expresion1
```

```
else
```

```
    expresion2
```

- **scripts:**

- **BEGIN:** commands are executed once only, BEFORE the first input record is read

- **END:** commands are executed once only, AFTER all the input is read

- **{ }** between BEGIN și END: commands are executed for each input record

- **examples:**

```
awk 'BEGIN{FS = ":"}' /etc/passwd
```

```
awk 'BEGIN{FS = ":"; OFS="\t"} {print $1, $2}' /etc/passwd
```

```
awk '/Ion/{cnt++}END{print "Ion apare de " cnt " ori."}' angajati.txt
awk 'END{print "Nr. angajati: " NR}' angajati.txt
awk 'BEGIN{total=0} {total++} END{print "Total: " total}' angajati.txt
```

- **instructions:**

<http://www.grymoire.com/Unix/AwkRef.html>

- **built-in functions:**

<http://www.grymoire.com/Unix/AwkRef.html>

REFERENCES:

- Regular expressions: <https://www.regular-expressions.info/quickstart.html>
- awk manual: <https://linux.die.net/man/1/awk>
- awk tutorial: <http://www.grymoire.com/Unix/Awk.html>
- grep manual: <https://linux.die.net/man/1/grep>
- sed manual: <https://linux.die.net/man/1/sed>
- sed tutorial: <http://www.grymoire.com/Unix/Sed.html#uh-41>

angajati.txt

Ionel Popescu 10/3/1961:Colinei,2,Cluj-Napoca:0740-123456 3500
Vasile Georgescu 5/10/1942:Piata Republicii,35,Cluj-Napoca:0722-654321 2850
Alexandru Ionescu 3/7/1971:Aleea Bibliotecii,10,Cluj-Napoca:0721-124536 3875
Tudor Alexandrescu 2/5/1963:Aleea Baisoara,53,Cluj-Napoca:0742-235641 2355
Victor Baci 25/9/1968:Eroilor,105,Floresti:0723-162453 4560
Horatiu Vasilescu 23/4/1965:Piata Marasti,13,Cluj-Napoca:0741-485769 37005
Adrian Pinte 11/8/1957:Lacrimioarelor,22,Cluj-Napoca:0742-258369 1942
Mircea Diaconu 6/11/1946:Prieteniei,7,Someseni:0744-147258 2565
Ovidiu Moldovan 17/1/1942:Almasului,65,Cluj-Napoca:0722-123789 1968
Puiu Calinescu 21/6/1920:Pitesti,88,Cluj-Napoca:0723-452163 1971
Olga Tudorache 24/1/1932:Florilor,41,Floresti:0744-458712 1942
Stela Enache 28/2/1952:Sindicatelor,75,Cluj-Napoca:0745-563214 1946
Radu Beligan 8/4/1949:Zambilei,98,Someseni:0744-852369 1957
Octavian Cotescu 17/12/1954:Stejarului,68,Floresti:0745-789456 32150
Silviu Achim 19/10/1936:Tudor Vladimirescu,18,Cluj-Napoca:0726-369147 1932
Toma Voicu 27/5/1948:Sportului,43,Floresti:0740-987125 1949
Ilarion Ciobanu 4/7/1931:Xenopol,32,Cluj-Napoca:0728-456987 1946
Gheorghe Dinica 30/2/1934:Vrabiilor,6,Someseni:0740-256314 1963
Liviu Ciulei 26/9/1947:Maramuresului,43,Cluj-Napoca:0741-785469 1920
Victor Rebengiuc 31/3/1931:Paris,9,Cluj-Napoca:0723-254136 1954
Vlad Nicolaescu 13/3/1965:Oasului,15,Cluj-Napoca:0745-741289 1949
