Princípios da Computação

The Unix shell 3

Regular expressions

Based on "Chapter 20: Regular Expressions" from
"*Linux Command Line and Shell Scripting Bible, 3rd Edition*"
Richard Blum and Christine Bresnahan. 2015, John Wiley & Sons, Inc.

isep Instituto Superior de Engenharia do Porto

What is a regular expression?

- A regular expression (*regex*) is a pattern that describes a set of strings that match the pattern.
- Typically used to find expressions in text.
  - Example: search for **cat**
  - Matches: **cat**, **cat**s, con**cat**enate, advo**cat**e

How do you build a regular expression?

- A regular expression pattern makes use of:
    - (static) characters and,
    - wildcard characters to represent unknown/variable data.

- You have already used wildcards with the **`ls`** command: *e.g.* * and ?

Types of regular expressions

- Different applications use different types of regular expressions.
- A *regular expression engine* is the software that interprets regular expression patterns.

Unix regular expression engines

- Basic Regular Expression (BRE)
    - Fast, good for most of the tasks.
- Extended Regular Expression (ERE)
    - Advanced pattern symbols, not so fast (?).

grep

- We will use the grep utility to search for regular expressions.
- The **`grep`** supports BRE.
- The **`egrep`** supports ERE.
    - … or, use the **`grep`** with the **`–E`** option.

BRE patterns

Plain text

- String of regular characters that represents the text to be searched.

- CaSe SeNsItIvE!!!!!!

- Examples:

  - **book** -> **book**shelf, **book**s

  - **one** -> **one**, Stall**one**

Plain text

- The grep command displays the lines where it finds a match to the regular expression.

```
$ echo 'I have three cats!' | grep at
I have three cats!
$ echo 'I have three dogs!' | grep at
$ echo 'I'm chatting with my friends.' | grep at
I'm chatting with my friends.
$
```

Anchor characters: start of line

- Start at the beginning of the line: ^
    - Defines a pattern starting at the beginning of the line.

```
$ echo 'Number 13' | grep ^1
$ echo '13 is the number' | grep ^1
13 is the number
$
```

Anchor characters: end of line

- Pattern at the end of the line: **$**
    - Defines a pattern at the end of the line.

```
$ echo 'Number 13' | grep 3$
Number 13
$ echo '13 is the number' | grep 3$
$
```

Any single character

- Single character: **.**
    - Matches any single character except a newline character.
    - If there is no character in the place of the dot, then the pattern fails.

Any single character

```
$ echo 'The drone is flying.' | grep 'ro.e'
The drone is flying.
$ echo 'I need a piece of rope!' | grep 'ro.e'
I need a piece of rope!
$ echo 'There are zero elements.' | grep 'ro.e'
There are zero elements.
$ echo 'There is John McEnroe.' | grep 'ro.e'
$
```


Does not match pattern:
r + o + *any-single-char* + e

Character classes

- Single character from a group (class) of characters: **[ ]**
    - Matches a single character from the group defined inside square brackets.
    - The brackets should contain any character you want to include in the class.

Character classes

```
$ echo 'Cat is an animal.' | grep '[BbCc]at'
Cat is an animal.
$ echo 'A bat can fly.' | grep '[BbCc]at'
A bat can fly.
$ echo 'My hat is on the chair.' | grep '[BbCc]at'
$ echo 'Combat boots' | grep '[BbCc]at'
Combat boots
$
```

Character classes - negating a class

- The caret **^** negates the character class: **[^]**

```
$ echo 'A bat can fly.' | grep '[^CcBb]at'
$ echo 'My hat is on the chair.' | grep '[^CcBb]at'
My hat is on the chair.
$
```

Character classes - ranges

- The dash **–** defines a range: **[–]**

- The range includes all characters that extend between both ends.

```
$ echo 'A bat can fly.' | grep '[c–mC–M]at'
$ echo 'Hat trick!' | grep '[c–mC–M]at'
Hat trick!
$
```

Special character classes

- There are pre-defined character classes:

[[:alpha:]]	Matches any alphabetical character, either upper or lower case
[[:alnum:]]	Matches any alphanumeric character 0–9, A–Z, or a–z
[[:blank:]]	Matches a space or Tab character
[[:digit:]]	Matches a numerical digit from 0 through 9
[[:lower:]]	Matches any lowercase alphabetical character a–z
[[:print:]]	Matches any printable character
[[:punct:]]	Matches a punctuation character
[[:space:]]	Matches any whitespace character: space, Tab, NL, FF, VT, CR
[[:upper:]]	Matches any uppercase alphabetical character A–Z

Special character classes

```
$ echo 'R2-D2 is a droid.' | grep '[[:upper:]][[:digit:]]-'
R2-D2 is a droid.
$ echo 'C-3PO is a droid.' | grep '[[:upper:]][[:digit:]]-'
$ echo 'R5-D5 was a jedi.' | grep '[[:upper:]][[:digit:]]-'
R5-D5 was a jedi.
$
```

Repeating character

- The asterisk * after a character means the character must appear zero or more times in the text.

```
$ echo 'neighbour (UK)' | grep 'bou*r'
neighbour (UK)
$ echo 'neighbor (US)' | grep 'bou*r'
neighbor (US)
$ echo 'neighbouuur (oops)' | grep 'bou*r'
neighbouuur (oops)
$
```

ERE patterns

Using ERE regular expressions

- To use ERE regular expressions, you need to:
  - use the **`egrep`** utility, or
  - add the **`-E`** option to the **`grep`** utility.

Non-repeating character

- The question mark **?** after a character means the character must appear zero or one time in the text.

```
$ echo 'neighbour (UK)' | egrep 'bou?r'
neighbour (UK)
$ echo 'neighbor (US)' | grep -E 'bou?r'
neighbor (US)
$ echo 'neighbouuur (oops)' | grep -E 'bou?r'
$
```

Repeating character

- The plus sign **+** after a character means the character must appear one or more times in the text.

```
$ echo 'neighbour (UK)' | egrep 'bou+r'
neighbour (UK)
$ echo 'neighbor (US)' | grep -E 'bou+r'
$ echo 'neighbouuur (oops)' | grep -E 'bou+r'
neighbouuur (oops)
$
```

Intervals

- You can specify how many times a character appears in a pattern by setting the limits (min and max) inside curly braces **{}** — the *interval*.

  - {*n*} : the regular expression appears exactly *n* times.

  - {*m*,*n*} : the regular expression appears at least *m* times, and no more than *n* times.

Intervals

```
$ echo 'neighbour (UK)' | egrep 'bou{0,2}r'
neighbour (UK)
$ echo 'neighbor (US)' | grep -E 'bou{0,2}r'
neighbor (US)
$ echo 'neighbouuur (oops)' | grep -E 'bou{0,2}r'
$
```

```
$ echo 'Phone: 900 234 876' | grep -E '[0-9]{3} [0-9]{3} [0-9]{3}'
Phone: 900 234 876
$
```

Alternative patterns

- The pipe | allows to specify two or more alternatives, making the regex engine to use a logical OR.

```
$ echo 'I like cats.' | grep -E 'cat|dog'
I like cats.
$ echo 'I love dogs.' | grep -E 'cat|dog'
I love dogs.
$
```

Grouping expressions

- Parenthesis **( )** allow to group an expression that is treated as a character.

```
$ echo "I don't like Mondays." | egrep '(Mon|Fri)day'
I don't like Mondays.
$ echo "I'll see you Thursday." | egrep '(Mon|Fri)day'
$ echo "Finally it's Friday\!" | egrep '(Mon|Fri)day'
Finally it's Friday!
$
```