Modulo 2 - Sezione B
Espressioni Regolari

Laboratorio di Sistemi Operativi I
Anno Accademico 2007-2008

Francesco Pedullà
(Tecnologie Informatiche)

Massimo Verola
(Informatica)

Introduzione alle espressioni regolari

- **Un'espressione regolare:**
  - è una stringa di caratteri e metacaratteri
  - i metacaratteri sono caratteri speciali che vengono interpretati in modo "non letterale" dal meccanismo delle espressioni regolari
- **Le espressioni regolari (Regular Expression, RE) sono utilizzate per ricerche e manipolazioni di stringhe**
  - Utilizzate da molti comandi quali **`grep`**, **`awk`**, **`sed`**, etc.
- **Definizione: *match***
  - Diciamo che una RE fa *match* con una particolare stringa se è possibile generare la stringa a partire dalla RE
- **Nota: Unix/Linux wildcard e RE hanno metacaratteri (e significati) differenti; non vanno confuse**

Espressioni regolari (I)

- **Sintassi:**
  - L'asterisco * fa match con qualsiasi numero di ripetizioni del carattere che lo precede, incluso 0
    - Esempio: "11*33" fa match con 1133, 11133, 111133, etc
  - Il punto . fa match con qualsiasi carattere, a parte newline
    - Esempio: "13.3" fa match con 13a3, 1303, ma non con 13\n3
  - Il caret ^ fa match con l'inizio di una linea (ma ha anche significati addizionali)
    - Esempio: "`^Subject:.*`" fa match con una linea di subject di posta elettronica
  - Il dollaro $ fa match con la fine di una linea:
    - Esempio: "`^$`" fa match con una linea vuota

Espressioni regolari (II)

- **Sintassi:**
  - Le parentesi quadre [...] sono utilizzate per fare match di un sottoinsieme (o di un intervallo) di caratteri
    - "[xyz]" fa match con i caratteri x, y, z
    - "[c-n]" fa match con qualsiasi carattere fra c ed n (nell'ordinamento ASCII)
    - "[a-zA-Z0-9]" fa match con qualsiasi caratter alfanumerico
    - "[0-9]*" fa match con qualsiasi stringa decimale
    - "[^0-9]" fa match con qualsiasi carattere non numerico
    - "[Yy][Ee][Ss]" fa match con yes, Yes, YES, yEs, etc.
- **Il backslash \ è usato come escape per i metacaratteri ed il carattere viene interpretato letteralmente: ad es., "\$"**
- **I metacaratteri perdono il loro significato speciale dentro [ ]**

Espressioni regolari (III)

- **Le espressioni regolari hanno un concetto di "parola":**
  - una parola è un pattern contenente solo lettere, numeri e underscore _
- **E' possibile fare matching**
  - con l'inizio di una parola: **\<**
    - Esempio: **\<Fo** fa match con tutte le parole che iniziano con Fo
  - con la fine di una parola: **\>**
    - Esempio: **ox\>** fa match con tutte le parole che finiscano con ox
  - con parole complete:
    - Esempio: **\<Fox\>**

Esempio: come cercare la parola **The** o **the** ?
RE #1: [tT]he ---> errata: troverebbe anche **other**
RE #2: □the□ ---> (□ *è blank*) errata: non troverebbe **the** all'inizio o fine di linea
RE #3: \<[tT]he\> ---> OK: utilizza la sintassi \<pattern\>

Espressioni regolari (IV)

- **RE Recall**
  - un modo per riferirsi ai match più recente
- **Sintassi:**
  - per "marcare" una porzione di espressione regolare che volete riutilizzare: racchiuderla in `\(  \)`
  - per ripetere una porzione "marcata", si può utilizzare \n, con n=1..9
- **Esempi:**
  - `'^\([a-z]\)\1'`
    - la sintassi `\([a-z]\)` indica un pattern [a-z] che, una volta fatto il match, puo' essere indicato con la sintassi \1
    - quindi la RE fa match con tutte le linee che iniziano con due lettere minuscole identiche
  - Ricerca di parole palindrome di 5 lettere (esempio: radar): `'\([a-z]\)\([a-z]\)[a-z]\2\1'`

Comando `grep`

- **Il comando `grep` permette di cercare pattern in un insieme di file**
  - se non vi sono file specificati, la ricerca è nello standard input
  - il pattern è espresso come espressione regolare
  - tutte le linee che contengono il pattern vengono stampate su stdout
  - in caso di file multipli, le linee sono precedute dal pathname del file che le contiene (a meno di opzione **`-h`**)
  - opzione **`-v`**
    - fa in modo che l'output contenga tutte le linee che non contengono il pattern
  - opzione **`-R`**
    - analizza ricorsivamente le subdirectory
  - opzione **`-c`**
    - conta le occorrenze, invece di stamparle
  - opzione **`-i`**
    - ignora distinzioni tra caratteri maiuscoli e minuscoli
  - opzione **`-q`**
    - modo "silente" (*quiet*), non scrive nulla su stdout

Comando `grep`

- **Esistono quattro versioni:**
  - `grep [options] pattern {file}*`
    - pattern è una espressione regolare
  - `fgrep [options] pattern [file(s)]`
    - pattern è una stringa fissa
    - versione più veloce
  - `egrep [options] pattern [file(s)]`
    - pattern è una espressione regolare estesa
  - `zgrep [options] pattern [file(s)]`
    - è anche in grado di cercare in file compressi (compressed or gzipped)

Comando grep - Esempi

```
if echo "$VAR" | grep -q txt ; then
   echo "$VAR contains the substring \"txt\""
fi
```

```
grep '[Ff]irst' *.txt
file1.txt:This is the first line of file1.txt.
file2.txt:This is the First line of file2.txt.
```

Esempi: cosa fanno queste ricerche?

- `grep -c 'ing$' /usr/dict/words`
- `grep -c '^un.*$' /usr/dict/words`
- `grep -ic '^[aeiou]' /usr/dict/words`
- `grep -ic '\(.\)\1\1' /usr/dict/words`
- `grep -c '^\(..\).*\1$' /usr/dict/words`
- `grep -ic '^\(.\)\(.\).\2\1$' /usr/dict/words`

Esercizi

- **Per verificare le soluzioni degli esercizi, scrivere un file contenente parole (una per linea) che possono fare match oppure no con la RE ed analizzarlo con `grep`.**
- **Definire una o più RE che facciano match con:**
  - qualunque stringa che contenga "a" o "b" seguito da 2 caratteri qualunque seguiti da "a" o "b". Le stringhe "axxb", "alfa" e "blka" fanno match, "ab" no.
  - una "A" seguita da qualunque carattere eccetto "x", "y" o "z".
  - qualunque numero intero composto da 5 digit.
  - qualunque parola (cioé una sequenza di caratteri alfanumerici senza spazi) che contenga una lettera doppia, per esempio “lettera” (doppia 't') o “doppia” (doppia 'p') – richiede 'recall', vedi lucido precedente

Espressioni regolari estese

- Il segno ? fa match con 0 o 1 ripetizioni della espressione regolare precedente
  - Esempio: “cort?o” fa match con coro e corto
- Il segno + fa match con 1 o più ripetizioni della espressione regolare precedente, ma non 0
  - Esempio: “ [0-9]+” fa match numeri non vuoti
- I segni \{ \} indicano il numero di ripetizioni della espressione regolare precedente
  - Esempio: “[0-9] \{ 5\}” fa il match con tutti i numeri a 5 cifre
- I segni () servono a raggruppare espressioni regolari
  - Esempio: “(re)*” fa match con “”, re, rere, rerere, etc.
- Il segno | indica un’alternativa (or)
  - Esempio: “(bio|psico)logia” fa match con biologia e psicologia

Esercizi

- **Per verificare le soluzioni degli esercizi, scrivere un file contenente parole (una per linea) che possono fare match oppure no con la RE ed analizzarlo con `grep`.**
- **Definire una o più RE che facciano match con:**
  - qualunque numero intero composto da 12 digit.
  - qualunque stringa che contiene solo 'a' e 'b'.
  - qualunque stringa composta di 'a' e 'b' che comincia e finisce con 'a'
  - qualunque stringa composta di 'a' e 'b' che contenga la stringa 'abba'

Sommario RE (I)

Operator	*Effect*
.	Matches any single character.
?	The preceding item is optional and will be matched, at most, once.
*	The preceding item will be matched zero or more times.
+	The preceding item will be matched one or more times.
\{N\}	The preceding item is matched exactly N times.
\{N,\}	The preceding item is matched N or more times.
\{N,M\}	The preceding item is matched at least N times, but not more than M times.
–	Represents the range if it's not first or last in a list or the ending point of range in a list.
^	Matches the empty string at the beginning of a line; also represents the characters not in the range of a list.
$	Matches the empty string at the end of a line.
\b	Matches the empty string at the edge of a word.
\B	Matches the empty string provided it's not at the edge of a word.
\<	Match the empty string at the beginning of word.
\>	Match the empty string at the end of word.

Sommario RE (II)

Pattern	*Matches*
^A	An A at the beginning of a line
A$	An A at the end of a line
A^	An A^ anywhere on a line
$A	A $A anywhere on a line
^\^	A ^ at the beginning of a line
^^	Same as ^\^
\$$	A $ at the end of a line
$$	Same as \$$
^.$	A line with any single character

Note that the caret ^ is only an anchor if it is the first character in a RE, while the $ is only an anchor if it is the last character in a RE.

Sommario RE (III)

Pattern	*Matches*
[0-9]	Any digit
[^0-9]	Any character other than a digit
[-0-9]	Any digit or a -
[0-9-]	Any digit or a -
[^-0-9]	Any character except a digit or a -
[ ]0-9]	Any digit or a]
[0-9]]	Any digit followed by a]
[0-9m-z]	Any digit or any character between m and z
[ ]0-9-]	Any digit, a -, or a]

Note that right square bracket] and dash – do not have special meaning if they directly follow a [.

Sommario RE (IV)

RE	*Matches*
*	Any line with a *
*	Any line with a *
\\	Any line with a \
^*	Any line starting with a *
^A*	Any line (useless RE !)
^A*	Any line starting with an A *
^AA*	Any line starting with one A
^AA*B	Any line starting with one or more A's followed by a B
^A\{4,8\}B	Any line starting with four, five, six, seven, or eight A's followed by a B
^A\{4,\}B	Any line starting with four or more A's followed by a B
^A\{4\}B	Any line starting with an AAAAB
\{4,8\}	Any line with a {4,8}
A{4,8}	Any line with an A{4,8}

*Note that modifiers like * and \{1,5\} only act as modifiers if they follow a character set.*

Sommario RE (V)

Hints for designing and analyzing a RE:

1. Knowing what it is you want to match and how it might appear in the text
2. Writing a pattern to describe what you want to match
3. Testing the pattern to see what it matches

Hints for evaluating the results of a pattern-matching operation:

1. ***Hits***: the lines I wanted to match
2. ***Misses***: the lines I didn't want to match
3. ***Misses that should be hits***: the lines that I didn't match but wanted to match
4. ***Hits that should be misses***: the lines that I matched but I didn't want to match