Regular Expressions in ambiente `vi`

mobytrick

8 settembre 2014

Ci si propone di illustrare l'uso delle *Regular Expression* congiuntamente con l'editor `vi`. Partiamo con l'allestimento di un file in cui il nome di 3 poeti è scritto in maniera piuttosto burocratica.

```
Alighieri Dante␣
Boccaccio␣␣␣Giovanni
␣Petrarca Francesco
```

Partendo dal presupposto che ogni riga comprenda 2 sole entità -cognome e nome- e che ciscuna di questa sia costituita da una sola parola, con l'uso di un solo comando è possibile riscrivere i nomi nella maniera più usuale, ovvero nome e cognome. Nel contempo vengono rimossi:

- gli eventuali spazi iniziali e finali
- gli eventuali spazi in eccesso tra le due componenti

È chiaro che si tratta di isolare le due entità. Vanno memorizzate in qualche buffer interno e poi ricomposte nell'ordine voluto. Si fa ricorso al meccanismo delle Regular Expressions noto come *backreference* che permette la memorizzazione di quanto conforme al modello in un buffer interno ed il suo successivo riutilizzo. Visto che le entità sono costituite da sole lettere e di queste la prima è maiuscola mentre tutto il resto è minuscolo, la Regular Expression che fa al caso è:

```
[A-Z][a-z]+
```

che però **non** memorizza alcunché. Si tratta di scriverla in modo da essere conforme al backreferencing. Viene pertanto formulata nel reguente modo:

```
\([A-Z][a-z]\+\)
```

ovvero: affinché quanto intercettato venga memorizzato in un buffer, il modello va racchiuso tra **parentesi tonde**, precedute dall'*escape* (il carattere \). Pure il metacarattere `+` va preceduto dall'escape. Se così non fosse, verrebbe considerato in maniera letterale. I normali motori che gestiscono le Regular Expressions (*RE engines*) di buffer interni ne mettono a disposizione 9, numerati dall'1 al 9. In una Regular Expression ci possono essere più sottostringhe che sfruttano la possibilità di memorizzare i caratteri intercettati. La prima di tali sottostringhe memorizza i caratteri nel buffer 1, la seconda nel 2 e via discorrendo. Per utilizzare il contenuto del buffer `n`, il numero va preceduto dal carattere escape.
Va da sé che l'uso del backreferencing consuma ingenti risorse computazionali. Operando da postazioni non particolarmente veloci, la macchina potrebbe apparire lenta.

Il comando `vi` per modificare una stringa nell'intero file è il seguente:

```
%s/old/new/
```

Mettendo a posto i vari tasselli, nel caso specifico la parte `old` del comando è formata:

```
␣*\([A-Z][a-z]\+\)␣\+\([A-Z][a-z]\+\)␣*
```

La Regular Expression è piuttosto lunga. A scopo didattico viene spezzata nei suoi componenti:

`␣*`	eventuali spazi in testa
`\([A-Z][a-z]\+\)`	la prima stringa (= cognome) nel buffer 1
`␣\+`	gli spazi centrali
`\([A-Z][a-z]\+\)`	la seconda stringa (= nome) nel buffer 2
`␣*`	eventuali spazi in coda

Tenuto conto delle memorizzazioni, la parte `new` del comando è così formata:

```
\2 \1
```

ovvero scrivi il contenuto del buffer 2 (nome), uno spazio bianco e poi quello del buffer 1 (cognome). In definitiva il comando, piuttosto lungo, è il seguente:

```
%s/ *\([A-Z][a-z]\+\) \+\([A-Z][a-z]\+\) */\2 \1/
```

C'è il rischio di sbagliare qualcosa. Prima di commettere danni si ricorre alla possibilità offerta da `vi` di scrivere i comandi a parte e farli eseguire in modalità *batch*. In caso di errori è sempre possibile metter mano al comando apportando le opportune modifiche. Supposto che il file su cui intervenire si chiami `testo.txt` ed il file dei comandi si chiami `comandi.txt`, si opera in questo modo:

```
vi -S comandi.txt testo.txt
```

I comandi vengono eseguiti e si rimane con la schermata finale che mostra il file `testo.txt` modificato. Se tutto è in ordine si possono rendere definitive le modifiche, altrimenti si esce lasciando il file intatto. Quando si è sicuri dell'assenza di errori, allora nel file dei comandi basta aggiugere come ultima riga `wq` ovvero `write & quit`. Se l'editor viene usato all'interno di uno script, per evitare sfarfallamenti dello schermo si usi la variante:

```
vi -S comandi.txt testo.txt 1> /dev/null 2>&1
```

ovvero output ed errori dirottati nella *pattumiera Unix*.