

# Introduzione alle espressioni regolari

## Introduzione

- Le espressioni regolari possono essere trovate in molti editor avanzati come il vi, nei programmi grep/egrep e in linguaggi come l'awk, il perl e il sed.
- Le espressioni regolari sono utilizzate per ricerche avanzate sensibili al contesto e per la revisione di testo.
- Un'espressione regolare è una descrizione formale di un modello che deve venir confrontato con una stringa di testo
  - Revisioni di testo e ricerche che normalmente richiederebbero ore (se fatte a mano) possono essere svolte in pochi secondi

## Introduzione

- Anche se le espressioni regolari sono molte diffuse nel mondo Unix non esiste nulla del tipo "il linguaggio standard delle espressioni regolari"
  - E più come se ci fossero molti dialetti diversi
- Per esempio ci sono due tipi di programmi grep: grep ed egrep.
  - Entrambi usano le espressioni regolari, ma con possibilità leggermente diverse.
- Il perl ha probabilmente la serie più completa di espressioni regolari.
- Fortunatamente tutti seguono gli stessi principi. Una volta compresa l'idea di base, è facile imparare i dettagli delle singole varianti.

## Esempio

- Supponiamo di avere a disposizione l'agenda telefonica di una di una ditta, strutturata nel seguente modo:

Phone	Name	ID
...	...	...
3412	Bob	123
3834	Jonny	333
1248	Kate	634
1423	Tony	567
2567	Peter	435
3567	Alice	535
1548	Kerry	534

...

## Esempio

- È una compagnia con 500 dipendenti.
- Tengono i dati in un semplice file `phonelist.txt`.
  - Le persone con 1 come prima cifra del numero di telefono lavorano nell'edificio 1.
- Chi lavora nell'edificio 1?
- La soluzione con le espressioni regolari può essere:  
`grep '^1' phonelist.txt`  
oppure:  
`egrep '^1' phonelist.txt`  
oppure:  
`perl -ne 'print if (/^1/)' phonelist.txt`

## Esempio

- Il segno `"^"` indica l'inizio di una riga
- Costringe l'intera espressione a "rispondere" solo se una linea ha un 1 come primo carattere

## Le regole di sintassi

- Pattern a carattere singolo:
  - La struttura base di un espressione regolare è il pattern a carattere singolo. Ricerca solo questo carattere. Un esempio di pattern a carattere singolo è l'1 nell'esempio precedente. Ricerca solo un 1 nel testo.
- Un altro esempio di pattern a carattere singolo è: `egrep 'Kerry' phonelist.txt` Questo pattern consiste solo di singoli caratteri (le lettere K,e ...)

## Set di caratteri

- I caratteri possono essere raccolti insieme in un set.
- Un set è rappresentato da una parentesi aperta ed una chiusa e da una lista di caratteri.
- Un set rappresenta in sé un unico carattere singolo. Uno ed uno solo di questi caratteri deve essere presente nel testo analizzato per far "reagire" il pattern.
- Per esempio:
  - `[abc]` è un pattern a carattere singolo che riconosce le lettere a, b o c

## Set di caratteri

- `[ab0-9]` è un pattern a carattere singolo che riconosce a, b o un numero nel range ASCII da zero a nove
- `[a-zA-Z0-9\_-]` riconosce un singolo carattere che è o una lettera maiuscola o minuscola, o un numero o il segno meno.
- Esempio: `egrep '^1[348]' phonelist.txt`
- Ricerca le righe che iniziano con 13, 14 o 18.

## Regole di sintassi

- Alcuni caratteri ASCII corrispondono solo a quel carattere mentre altri hanno un significato particolare.
- Per esempio la parentesi quadra inizia un set.
- Nel set `"-"` ha il significato particolare di range.
- Per eliminare il significato particolare potete precedere il carattere con un backslash `"\"`.
- Il segno meno in `[a-zA-Z0-9\_-]` è un esempio di ciò.

## Il carattere '.'

- Il punto è un carattere speciale importante. Riconosce tutto ad eccezione del carattere di cambio riga.
- Per esempio:  
grep '^.2' phonelist.txt  
oppure:  
egrep '^.2' phonelist.txt  
ricerca le linee con un 2 in seconda posizione ed un qualsiasi come primo carattere.

## Set invertiti

- I set possono essere invertiti iniziandone la definizione con "[^" invece che con "[".
- Il segno "^" non significa più l'inizio della riga ma la combinazione di "[" e "^" indica il set invertito.
- [0-9] è un pattern a carattere singolo che ricerca i numeri nel range ASCII da zero a nove.
- [^0-9] ricerca ogni carattere che non sia una cifra.
- [^abc] ricerca ogni carattere che non sia a, b o c.
- Il punto ricerca qualsiasi carattere fatta eccezione per il segno di cambio riga. È lo stesso che [^\n].
  - Dove \n e' il carattere di cambio riga.

## Set invertiti

- Per cercare tutte le righe che non iniziano con un 1 possiamo scrivere:  
grep '^[^1]' phonelist.txt  
oppure:  
egrep '^[^1]' phonelist.txt

## Ancore

- Già nella parte precedente abbiamo visto che "^" corrispondeva all'inizio riga.
- Le ancore sono speciali caratteri che corrispondono a posizioni nel testo e non a caratteri presenti nel testo stesso.
- ^ Corrisponde all'inizio di una riga
- \$ Corrisponde alla fine di una riga
- Per cercare una persona della compagnia con ID 567 nella lista phonelist.txt si può scrivere:  
egrep '567\$' phonelist.txt
- questo ricerca le righe con 567 a fine riga.

## Moltiplicatori

- Un moltiplicatore determina quante volte un pattern a singolo carattere deve essere presente nel testo.

descrizione	grep	egrep	perl	vi	vim / t h < < th > vile	elvis	emacs	
zero o più volte	*	*	*	*	*	*	*	*
una o più volte	\{1,\}	+	+		\+	\+	\+	+
zero o una volta	\?	?	?		\=	\?	\=	?
da n a m volte	\{n,m\}		{n,m}				\{n,m\}	\{n,m\}

## Esempio

- Per trovare una riga che inizia con un 1, ha qualche cifra, almeno uno spazio ed un nome che inizia per K possiamo scrivere:

```
grep '^1[0-9]\{1,\} \{1,\}K' phonelist.txt
```

o usare \* e ripetere [0-9] e lo spazio:

```
grep '^1[0-9][0-9]* *K' phonelist.txt
```

oppure:

```
egrep '^1[0-9]+ +K' phonelist.txt
```

oppure:

```
perl -ne 'print if (/^1[0-9]+ +K/)' phonelist.txt
```

## Moltiplicatori

- Il moltiplicatore moltiplica la presenza del pattern che lo precede. Quindi "23\*4" NON significa "2 poi 3 e non 4" (questo sarebbe "23.\*4"). Significa "una volta 2 poi forse molte volte 3 ed una 4"
- È anche importante notare che questi moltiplicatori sono "avid".
  - il primo moltiplicatore presente estende la sua influenza il più possibile.
- L'espressione `^1.*4` troverebbe l'intera riga:  
1548 Kerry 534  
dall'inizio fino all'ultimo 4. Non riconosce il solo 154.
- Questo non fa una gran differenza per il grep, ma è importante per le revisioni di testo e le sostituzioni.

## L'uso delle parentesi come memoria

- Le parentesi usate come memoria non cambiano il modo in cui un'espressione riconosce il testo ma invece fanno memorizzare il testo incluso tra esse, in modo che ci si possa riferire ad esso più avanti nell'espressione.
- La parte memorizzata è disponibile attraverso variabili. Il primo blocco memorizzato tra parentesi corrisponde alla variabile uno, il secondo alla due e così via.

nome programma	sintassi delle parentesi	sintassi delle variabili
grep	\(\)	\1
egrep	()	\1
perl	()	\1 o \${1}
vi,vim,vile,elvis	\(\)	\1
emacs	\(\)	\1

## Esempio

- L'espressione `[a-z][a-z]` riconoscerà due lettere minuscole. Si possono usare le variabili per ricercare pattern come 'otto':  
`egrep '([a-z])([a-z])\2\1'`
- La variabile `\1` contiene la lettera 'o' e la `\2` la lettera 't'.
- L'espressione riconoscerebbe anche il nome `anna` ma non `yxyx`.
- Le parentesi per la memorizzazione di blocchi non sono molto usate per la ricerca di nomi come `otto` od `anna`, ma piuttosto per le revisioni e le sostituzioni

## Uso delle espressioni regolari per la revisione del testo

- Per il lavoro di revisione bisogna usare un editor come il `vi` o `cs`, oppure usare, ad esempio, il `perl`.
- Gli esempi successivi faranno riferimento all'uso di `vi` dove il comando di sostituzione è  
`:%s/ / /g.`
- La percentuale si riferisce al range "tutto il file" e può essere sostituita da qualsiasi range appropriato (vedi manuale).
- Il comando `'gc'` è la versione interattiva. Quella non interattiva è `s/ / /g`
  - Interattivo significa che ad ogni ritrovamento `vi` viene chiesto se effettuare o meno la sostituzione

## Esempio

- Il modo di numerazione della compagnia è stato modificato, ed ad ogni numero che inizia con 1 deve essere aggiunto un 2 dopo la seconda cifra. Questo significa che, ad esempio, 1423 diventerà 14223.
- La vecchia lista:

Phone	Name	ID
...		
3412	Bob	123
3834	Jonny	333
1248	Kate	634
1423	Tony	567
2567	Peter	435
3567	Alice	535
1548	Kerry	534
...		

## Esempio

- Con il vi

```
:%s/^(1.)\12/g
```

Phone	Name	ID
...		
3412	Bob	123
3834	Jonny	333
12248	Kate	634
14223	Tony	567
2567	Peter	435
3567	Alice	535
15248	Kerry	534
...		

## Esempio

- Ora l'allineamento nella lista è stato alterato.
- Come si può risolvere questo problema?
- Si può verificare se vi è uno spazio bianco in quinta posizione ed inserirne un altro:  
`:%s/^\(.... \)\^1 /g`

## Esempio

- Un collega ha editato la lista manualmente e accidentalmente ha inserito qualche spazio all'inizio di alcune righe.

- Come possiamo eliminarli?

Phone	Name	ID	
...			
3412	Bob	123	
3834	Jonny		333
1248	Kate	634	
1423	Tony	567	
2567	Peter		435
3567	Alice	535	
1548	Kerry		534
...			

- Questo dovrebbe rimuoverli: `%s/^\s*/g`