

Shell

Espressioni regolari

Metacaratteri

La shell Unix riconosce alcuni caratteri speciali, chiamati **metacaratteri**, che possono comparire nei comandi.

Quando l'utente invia un comando, la shell lo scandisce alla ricerca di eventuali metacaratteri, che processa in modo speciale.

Una volta processati tutti i metacaratteri, viene eseguito il comando.

Esempio:

```
user> ls *.java
```

```
Albero.java          div.java             ProvaAlbero.java
AreaTriangolo.java  EasyIn.java         ProvaAlbero1.java
AreaTriangolo1.java IntQueue.java
```

Il **metacarattere** `*` all'interno di un pathname è un'**abbreviazione** per un nome di file. Il pathname `*.java` viene espanso dalla shell con tutti i nomi di file che terminano con l'estensione `.java`. Il comando `ls` fornisce quindi la lista di tutti

Abbreviazione pathname

I seguenti metacaratteri, chiamati **wildcard** sono usati per **abbreviare** il nome di un file in un pathname:

Metacarattere	Significato
*	stringa di 0 o più caratteri
?	singolo carattere
[]	singolo carattere tra quelli elencati
{ }	stringa tra quelle elencate

Esempi:

```
user> cp /JAVA/Area*.java /JAVA_backup
```

copia tutti i files il cui nome inizia con la stringa Area e termina con l'estensione .java nella directory JAVA_backup.

```
user> ls /dev/tty?
```

```
/dev/ttya /dev/ttyb
```

... esempi

```
user> ls /dev/tty?[234]
/dev/ttyp2 /dev/ttyp4 /dev/ttyq3 /dev/ttyr2 /dev/ttyr4
/dev/ttyp3 /dev/ttyq2 /dev/ttyq4 /dev/ttyr3
```

```
user> ls /dev/tty?[2-4]
/dev/ttyp2 /dev/ttyp4 /dev/ttyq3 /dev/ttyr2 /dev/ttyr4
/dev/ttyp3 /dev/ttyq2 /dev/ttyq4 /dev/ttyr3
```

```
user> mkdir /user/studenti/rossi/{bin,doc,lib}
crea le directory bin, doc, lib .
```

Quoting

Il meccanismo del **quoting** è utilizzato per inibire l'effetto dei metacaratteri. I metacaratteri a cui è applicato il quoting perdono il loro significato speciale e la shell li tratta come caratteri ordinari.

Ci sono tre meccanismi di quoting:

- il metacarattere di **escape** `\` inibisce l'effetto speciale del metacarattere che lo segue:

```
user> cp file file\  
user> ls file*  
file      file?
```

- tutti i metacaratteri presenti in una stringa racchiusa tra **singoli apici** perdono l'effetto speciale:

```
user> cat 'file*?'  
...
```

- i metacaratteri per l'abbreviazione del pathname presenti in una stringa racchiusa tra **doppi apici** perdono l'effetto speciale (ma non tutti i metacaratteri della shell):

```
user> cat "file*?"
```

Metacaratteri comuni di Shell

Simbolo	Significato	Esempio d'uso
>	Ridirezione dell'output	<code>ls >temp</code>
>>	Ridirezione dell'output (append)	<code>ls >>temp</code>
<	Ridirezione dell'input	<code>wc -l <text</code>
<<delim	ridirezione dell'input da linea di comando (here document)	<code>wc -l <<delim</code>
*	Wildcard: stringa di 0 o più caratteri, ad eccezione del punto (.)	<code>ls *.c</code>
?	Wildcard: un singolo carattere, ad eccezione del punto (.)	<code>ls ?.c</code>
[...]	Wildcard: un singolo carattere tra quelli elencati	<code>ls [a-zA-Z].bak</code>
{...}	Wildcard: le stringhe specificate all'interno delle parentesi	<code>ls {prog,doc}*.txt</code>

Metacaratteri comuni

Simbolo	Significato	Esempio d'uso
	Pipe	<code>ls more</code>
;	Sequenza di comandi	<code>pwd;ls;cd</code>
	Esecuzione condizionale. Esegue un comando se il precedente fallisce.	<code>cc prog.c echo errore</code>
&&	Esecuzione condizionale. Esegue un comando se il precedente termina con successo.	<code>cc prog.c && a.out</code>
(...)	Raggruppamento di comandi	<code>(date;ls;pwd)>out.txt</code>
#	Introduce un commento	<code>ls # lista di file</code>
\	Fa in modo che la shell non interpreti in modo speciale il carattere che segue.	<code>ls file.*</code>
!	Ripetizione di comandi memorizzati nell'history list	<code>!ls</code>

Comandi di Filtro

I **filtri** sono una particolare classe di comandi che possiedono i seguenti requisiti:

- prendono l'input dallo **standard input device**,
- effettuano delle operazioni sull'input ricevuto,
- inviano il risultato delle operazioni allo **standard output device**.

Tali comandi risultano quindi degli ottimi strumenti per costruire pipeline che svolgano compiti complessi.

Ad esempio:

```
> uniq file
```

restituisce in output il contenuto del file `file`, sostituendo le linee adiacenti uguali con un'unica linea.

grep [*opzioni*] *pattern* [*nomefile*]

- Stampa le righe del file che corrispondono al pattern
- Il pattern è una *espressione regolare*
 - nel caso più semplice, il pattern può essere una stringa senza caratteri speciali
 - `grep a pippo.txt` stampa le righe di `pippo.txt` che contengono una `a`
- Se *nomefile* non è specificato, legge da standard input
 - `ls -l | grep 2006` elenca i file che sono stati modificati l'ultima volta nel 2006 (oltre a...)
 - `ls -l | grep rwx` elenca i file per cui almeno una categoria di utenti ha tutti i permessi (oltre a...)

grep [*opzioni*] *pattern* [*nomefile*]

Alcune Opzioni:

- -v stampa le righe che *non* corrispondono al pattern
- -n l'output e' nel formato: <indice>:<riga>
 - dove <indice> corrisponde al numero di <riga> all'interno del file
- -c conta visualizza solo il numero di occorrenze della stringa nel file
- -i rende il comando “case-INsensitive”

grep: Esempi

```
lso:~>grep root /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
```

```
operator:x:11:0:operator:/root:/sbin/nologin
```

```
lso:~>grep -n root /etc/passwd
```

```
1:root:x:0:0:root:/root:/bin/bash
```

```
12:operator:x:11:0:operator:/root:/sbin/nologin
```

```
lso:~>grep -c root /etc/passwd
```

```
2
```

grep: Esempi

```
lso:~>grep -v bash /etc/passwd |grep -v nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
news:x:9:13:news:/etc/news:
```

```
lso:~>grep -c account /etc/passwd
```

```
0
```

```
lso:~>grep -c -i account /etc/passwd
```

```
1
```

```
lso:~>grep -i account /etc/passwd
```

```
lso:x:501:501:LSO Account:/home/lso:/bin/bash
```

```
lso:~>grep -i account /etc/passwd |wc -l
```

```
1
```

Espressioni Regolari: I caratteri

- Una espressione regolare e' un *pattern che descrive un insieme di stringhe*
- *L'elemento atomico delle espressioni regolari e' il carattere*
 - *Un carattere e' una espressione regolare che descrive se stesso*
 - *L'espressione "a" descrive "l'insieme di stringhe {a}"*
- *La maggior parte dei caratteri sono "espressioni regolari"*
- *I Metacaratteri corrispondono ad operatori*
 - *Un metacarattere puo' essere utilizzato con il suo valore utilizzando il carattere di escape "\"*

Composizione di E.R.

- "." qualunque carattere (1)
- exp* zero o più occorrenze di exp (2)
- ^exp exp all'inizio del rigo (1)
- exp\$ exp alla fine del rigo (1)
- [a-z] un carattere nell'intervallo specificato
- [^a-z] un carattere fuori dall'intervallo
- [aqwe] un carattere nell'insieme

Note: (1) è un carattere normale per Bash
(2) ha un significato diverso per Bash

Composizione di E.R.

- `\<exp` `exp` all'inizio di una parola
- `exp\>` `exp` alla fine di una parola
- `exp{N}` `exp` compare esattamente N volte
- `exp{N,}` `exp` compare almeno N volte
- `exp{N,M}` `exp` compare almeno N ed al piu' M volte
- `[[CLASS:]]` un carattere nella classe CLASS
 - CLASS puo' assumere valori come `digit`, `upper`, `lower`...

Esempi

- `a*b` zero o più a seguite da una b
- `a.*b` una a prima di una b
- `\<[[:upper:]]` una parola che inizia con lettera maiuscola
- `^d` la lettera d all'inizio del rigo
- `^a*$` un rigo vuoto o composto solo di a
- `^a.*b$` un rigo che inizia con a e finisce con b
- `\<.-` una parola che ha un trattino al secondo posto

Es.:

```
user> ls -l | grep ^-rwx
```

```
user> grep :.*e$ /etc/passwd
```

Shell ed Espressioni Regolari

Molti comandi per l'elaborazione di testi di UNIX (ad esempio [grep](#), [ed](#), [sed](#), ..) consentono la definizione di [espressioni regolari](#), ossia di [schemi per la ricerca di testo](#) basati sull'impiego di [metacaratteri](#):

- Generalmente, i metacaratteri usati da tali comandi [non](#) coincidono con i metacaratteri impiegati dalla shell per identificare i nomi dei file
- Molti caratteri che hanno un significato speciale nelle espressioni regolari [hanno pure](#) un significato speciale per la shell



- Attenzione a non confondere i metacaratteri di shell con quelli che non lo sono
- Utilizzare gli apici ' ' o i doppi apici " " per racchiudere le espressioni regolari

Composizione di E.R.

- La “concatenazione” di espressioni regolari e' una espressione regolare:
 - Le “stringhe” possono essere costruite dalla “concatenazione” dei caratteri.
 - Una stringa corrisponde (“match”) ad una concatenazione di stringhe se e' composta da due sottostringhe che corrispondono, rispettivamente, alle due espressioni regolari
 - “ab” corrisponde alla concatenazione di $exp1=“a”$ ed $exp2=“b”$
- L'operatore “|” (es. $exp3=exp1|exp2$)
 - Una stringa corrisponde ad $exp3$ se esiste un match con $exp1$ o con $exp2$.

Espressioni Regolari Estese

- exp^+ una o più occorrenze di exp (1)
- $exp?$ zero o una occorrenza di exp (2)
- $exp1 | exp2$ $exp1$ oppure $exp2$ (2)
- (exp) equivalente a exp , serve per stabilire l'ordine di valutazione

Nelle espressioni regolari “di base” i caratteri "?", "+", "{", "|", "(", e ")" devono essere preceduti dal carattere di escape “\”

Esempi

- `[[[:digit:]]\+` una sequenza non vuota di cifre
- `^a|b` un rigo che inizia con a oppure contiene b
- `^(a|b\)` un rigo che inizia con a oppure con b
- `\(\.txt\)|\(\.doc\)\>` una parola che termina con .txt o con .doc
(in `egrep`, `(\.txt)|\.doc`)

- In **grep**, questi simboli vanno preceduti da “\” (backslash)
- In **egrep** (*extended grep*), vanno usati direttamente

Esempio

```
lso:~>egrep '^r.*n$|^r.*37' /etc/passwd
```

```
rpm:x:37:37::/var/lib/rpm:/bin/bash
```

```
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
```

```
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
```

```
lso:~>grep '^r.*n$|^r.*37' /etc/passwd
```

```
lso:~>grep '^r.*n$|^r.*37' /etc/passwd
```

```
rpm:x:37:37::/var/lib/rpm:/bin/bash
```

```
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
```

```
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
```

Precedenza

- Ordine di precedenza:
 - Ripetizione
 - Concatenazione
 - “|”
- E' possibile utilizzare le parentesi tonde per ridefinire le precedenze
 - Esattamente come avviene per le espressioni aritmetiche.

Esempio

Iso:~>egrep '5|1:+' /etc/passwd

Corrisponde a tutte le stringhe che

- terminano con un “1” seguito da almeno una occorrenza di “:”
 - 1:, 1::, 1::: ...

OPPURE

- contengono “5”

Iso:~>egrep '(5|1):+' /etc/passwd

Corrisponde a tutte le stringhe che

- terminano con un “1” seguito da almeno una occorrenza di “:”
 - 1:, 1::, 1::: ...

OPPURE

- terminano con un “5” seguito da almeno una occorrenza di “:”
 - 5:, 5::, 5::: ...

Esempio

```
Iso:~>egrep '5|1:+' /etc/passwd
```

```
bin:x:1:1:bin:/bin:/sbin/nologin
```

```
sync:x:5:0:sync:/sbin:/bin/sync
```

```
operator:x:11:0:operator:/root:/sbin/nologin
```

```
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
```

```
nfsnobody:x:65534:65534:Anonymous NFS:/var/lib/nfs:/sbin/nologin
```

```
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
```

```
named:x:25:25:Named:/var/named:/sbin/nologin
```

```
Iso:x:501:501:LSO Account:/home/Iso:/bin/bash
```

```
Iso:~>egrep '(5|1):+' /etc/passwd
```

```
bin:x:1:1:bin:/bin:/sbin/nologin
```

```
sync:x:5:0:sync:/sbin:/bin/sync
```

```
operator:x:11:0:operator:/root:/sbin/nologin
```

```
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
```

```
named:x:25:25:Named:/var/named:/sbin/nologin
```

```
Iso:x:501:501:LSO Account:/home/Iso:/bin/bash
```

Esempio

```
Iso:~>egrep '(501:){2}' /etc/passwd
```

Ritorna tutte le righe che contengono due occorrenze consecutive della stringa 501:

```
Iso:~>egrep '501:{2}' /etc/passwd
```

Ritorna tutte le righe che contengono la stringa 501:: (due occorrenze di “:”)

Nota: Sono equivalenti a:

```
Iso:~>grep '\(501:\)\{2\}' /etc/passwd
```

```
Iso:~>grep '501:\{2\}' /etc/passwd
```

Esempio

```
Iso:~>egrep '(501:){2}' /etc/passwd
```

```
Iso:x:501:501:LSO Account:/home/Iso:/bin/bash
```

```
Iso:~>egrep '501:{2}' /etc/passwd
```

(nessun output)

Riferimenti

- Capitolo 4 di [Bash Guide for Beginners]