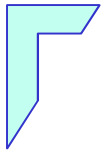
Lab. di Sistemi Operativi
- Lezione in aula -
a.a. 2012/2013

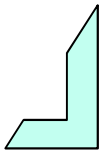
"Espressioni Regolari"

Sommario

- Metacaratteri
- Quoting
- Comandi di filtro:
  - grep
- Espressioni Regolari (E.R.):
  - I caratteri
  - grep con E.R.



- Metacaratteri -



Metacaratteri

- Sono **caratteri speciali** che possono essere **inseriti nei comandi** e quindi processati in maniera speciale dalla shell Unix prima di eseguire il comando vero e proprio.

- Esempio di metacarattere: * usato all'interno di un pathname serve ad abbreviare il nome di un file. Quindi il pathname *.java viene processato dalla shell come tutti i nomi dei file di estensione .java

Esempio d'uso:

$ ls *.java

fornisce la lista di tutti i file con estensione .java

Metacaratteri per abbreviare un 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
```

Visualizza il contenuto di ttya e di ttyb

Metacaratteri per abbreviare un pathname

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 -

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?
```

copia file in file? inibendo l'effetto del metacarattere ?

- 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*?"
```

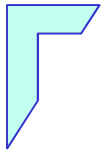
Metacaratteri comuni

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

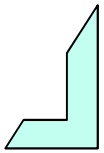
```
user> wc -l <<delim       # here document
? queste linee formano il contenuto
? del testo
? delim
2
```

Metacaratteri comuni

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



- Comandi di filtro -



Comandi di filtro

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

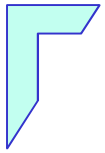
- prendono l'input dallo **standard input device**,
- effettuano delle operazioni sull'input ricevuto,
- inviano il risultato delle operazioni allo **standard output device**.

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

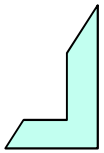
Ad esempio:

```
> uniq file
```

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



- Comandi di filtro: grep -



Comando di filtro: grep

- Comando grep:

Sintassi: 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

Esempio: grep a pippo.txt

- stampa le righe di pippo.txt che contengono una **a**

Esempi: ls – l | grep 2010 ls –l | grep rwx

- elenca i file che sono stati modificati l'ultima volta nel 2010
- elenca i file per cui almeno una categoria di utenti ha permessi di rwx

Comando di filtro: grep

- Comando grep alcune opzioni:

-v stampa le righe che non corrispondono al pattern (filtro)

-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

-i rende il comando "case-INsensitive"

Comando di ricerca: grep

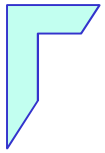
- Comando grep esempi:

```
lso:~>grep root /etc/passwd                    (senza opzione)
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

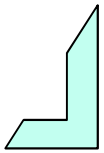
```
lso:~>grep -n root /etc/passwd                 (opzione -n)
1:root:x:0:0:root:/root:/bin/bash
12:operator:x:11:0:operator:/root:/sbin/nologin
```

```
lso:~>grep -c root /etc/passwd                 (opzione -c)
2
```

```
lso:~>grep -v bash /etc/passwd | grep -v nologin      (opzione -v)
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:
```



- Espressioni regolari: I caratteri -



Espressioni regolari: I caratteri

- Un' 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 può essere utilizzato con il suo valore utilizzando il carattere di escape "\"

Composizione di E.R.

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

E.R.: 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

Esempio:

user> ls -l | grep ^-rwx

visualizza tutti i file che hanno permessi di lettura scrittura ed esecuzione per l'utente corrente

Concatenazione 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

- Sono un' estensione dell' Espressioni regolari di base:
  - exp+ — una o più occorrenze di exp
  - exp? — zero o una occorrenza di exp
  - exp1 | exp2 — exp1 oppure exp2
  - (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 "\"

Espressioni Regolari Estese: 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

Sintassi per egrep:

egrep <regexpr> <filename>

E.R.: Esempi

- Esempi:

```
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

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 -

- E' possibile utilizzare le parentesi tonde per ridefinire le precedenze, come avviene per le espressioni aritmetiche.
- Esempi:

```
lso:~>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"

```
lso:~>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::: ...**

- Esempi -

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

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

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

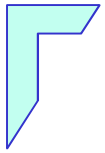
```
lso:~>egrep '501:{2}' /etc/passwd
```

- Ritorna tutte le righe che contengono la stringa 501:: (due occorrenze di ":")

- Equivalenti a:

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

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



- Fine Lezione -