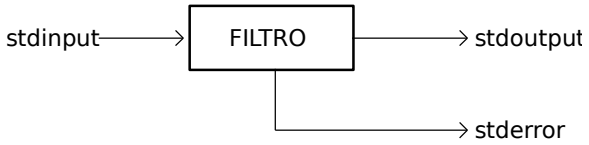
Lavorare coi file

Stream e ridirezione
Filtri di base
Ricerca di file
Archiviazione e compressione

Stream predefiniti

Si è detto che in unix qualsiasi cosa viene astratta come file. In particolare, qualsiasi comando unix ha a disposizione 3 file con cui comunicare con il resto del sistema:
standard input in ingresso
standard output in uscita
standard error in uscita



Stream predefiniti

Per i comandi lanciati da un terminale, lo standard input viene agganciato alla tastiera e gli altri due al video

Esempi:

Il comando cat riporta sullo standard output i caratteri letti dai file che vengono nominati come argomenti:

cat prova.c hello.c
cat senza argomenti preleva i caratteri dallo standard input e li riversa sullo standard output

Concatenazione di comandi

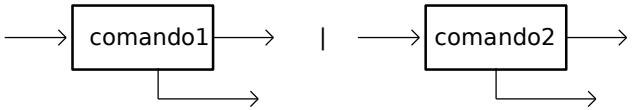
Il comando **less** riversa i caratteri provenienti dal suo standard input allo standard output arrestando l'operazione al raggiungimento della pagina (cioè dopo aver riempito un numero di righe pari all'altezza del terminale).

Il comando less, si comporta come un *filtro* che apporta modifiche al flusso di caratteri in ingresso per produrre un flusso di caratteri in uscita. In unix esistono diversi comandi che si comportano come filtri e la shell mette a disposizione un metodo per il collegamento dei vari flussi di caratteri: la **pipe**

L'indicazione dei collegamenti avviene sulla command line, intercalando ogni comando al successivo con il simbolo **|**

Concatenazione di comandi

Ad esempio, se si vuole utilizzare il filtro less sull'output del comando cat:
cat /etc/passwd | less



All'atto pratico la concatenazione fa apparire due comandi come uno solo, avente per STDIN quello del primo, STDOUT quello del secondo, e SDTERR la sovrapposizione dei due.
E' possibile comporre filtri comunque complessi concatenando più filtri fra loro:
ls | grep ^pro.a$ | less

Funzionamento della pipe

La pipe e' un oggetto messo a disposizione dal sistema operativo unix per collegare i flussi di dati di due processi. Il sistema si occupa della gestione e della sincronizzazione di questo sistema di comunicazione che rappresenta uno dei cosiddetti sistemi IPC.
A differenza della pipe fornita dal DOS, quella di unix non fa uso di file temporanei ma si appoggia interamente alle capacita' di buffering del sistema operativo che provvede a 'bloccare' il processo produttore di dati quando il processo ricevitore risulti piu' lento e, alternativamente, bloccare il processo ricevitore quando il produttore risulti piu' lento.

Funzionamento della pipe

Entrambi i processi, comunque, sono contemporaneamente attivi senza presentare la latenza tipica dei sistemi DOS. Questo permette la concatenazione di più comandi senza che l'output venga rinviato al completamento dell'ultimo.

Ovviamente, se un comando della pipeline deve operare delle modifiche su tutto lo stream di dati (es. ordinamento), allora l'output sarà disponibile solo al termine della lettura.

La pipe si puo' considerare una forma particolare di *ridirezione*. Il meccanismo della pipe esegue un collegamento (ridirezione) fra lo stdout del comando a sinistra della pipe con lo stdin del comando alla destra.

Funzionamento della pipe

Ogni comando presente nella pipeline viene eseguito da una subshell diversa, generata da una *fork*().

- Nel caso il comando sia esterno, sostitusce il processo *bash* in seguito alla chiamata di *exec*()
- Nel caso invece sia un built-in, è naturalmente la subshell medesima (quindi un processo *bash* figlio di quello che sta interpretando la pipeline nel suo complesso) ad attuarlo

Funzionamento della pipe con subshell

In questo secondo caso bisogna porre attenzione a non farsi trarre in inganno dall'apparente uniformità di risorse tra shell principale e subshell:

```
export COUNT=0; cat file | while read line ;
do COUNT=$[ $COUNT + 1 ] ; done ; echo $COUNT
```

Il ciclo evidenziato viene eseguito da una subshell, che riceve per copia l'environment della shell principale, aggiorna la sua copia della variabile COUNT… e termina, distruggendola.
Il valore della variabile COUNT della shell principale non viene alterato, e l'*echo* finale stampa 0.

Funzionamento della pipe con subshell

In questi casi è possibile forzare con **( )** l'esecuzione di una sequenza di comandi in una subshell: l'interprete in tal caso

- non ne aprirà di superflue per i interpretare i builtin
- chiuderà la subshell solo quando esplicitamente indicato

```
export COUNT=0; cat file | ( while read line ;
do COUNT=$[ $COUNT + 1 ] ; done ; echo
$COUNT )
```

In questo esempio la variabile COUNT aggiornata dal ciclo *while* è la stessa usata da *echo*, poiché entrambi i comandi sono eseguiti nella subshell eseguita grazie all'indicazione delle parentesi tonde

Ridirezione dell'output su file

E' possibile riversare i dati prodotti da un comando (ad esempio la lista prodotta da ls) su un file, utilizzando i simboli di ridirezione **>** e **>>**.

ls > miofile mette il risultato di ls nel file miofile, senza presentare nulla a video.

ls >> miofile esattamente come sopra, con la differenza che, se miofile esiste, allora l'output di ls viene aggiunto al termine del contenuto corrente

Ridirezione dell'error su file

Se si vogliono ridirigere solo i messaggi di errore si deve utilizzare il simbolo **2>** o **2>>**:

ls nomedifileinesistente 2> miofile

Lo stderr puo' anche essere ridiretto all'attuale stdout, con il simbolismo:

ls > miofile 2>&1

il quale ridirige lo stdout di ls a *miofile* ed i messaggi di errore di ls all'attuale stdout, quindi lo stesso *miofile*.

Ridirezione dell'error su pipe

Per inviare sia l'output che l'error di un comando all'input di un altro, tramite una pipe, si usa:

comando1 2>&1 | comando2

Non esiste un operatore per selezionare direttamente il solo standard error al fine di inviarlo ad una pipe. Si può usare però un accorgimento:

comando1 2>&1 > file_destinazione_output | comando2

In questo modo ciò che esce da stderr viene inviato al file descriptor 1, poi intercettato dalla pipe, e ciò che esce da stdout viene inviato a file_destinazione_output.

Ridirezione dell'input da file

E' possibile anche utilizzare un file come sorgente di dati riversandone il contenuto sullo stdin di un comando:

more < miofile utilizza il contenuto di miofile come inpout al comando more.

Ricerca di parti in un testo: Grep

grep è forse la più nota delle utility.

Esamina le righe del testo in ingresso (su standard input o specificato come elenco di file sulla riga di comando), e riproduce in uscita quelle che contengono l'*espressione regolare* (nel caso più semplice una sottostringa) passata come argomento. Es. per cercare un file di nome "prova" all'interno dell'output di ls:

ls | grep prova

Da qui in avanti faremo riferimento alla variante di grep che supporta le espressioni regolari "moderne": egrep

Espressioni regolari moderne (o estese)

L'utility egrep (ma non solo) utilizza una sintassi per la specifica dei *pattern* di ricerca detta espressione regolare (regexp o RE). La documentazione è reperibile in regex(7).

In sintesi

- RE = uno o più rami non vuoti separati da |
- ramo = uno o più pezzi concatenati tra loro
- pezzo = atomo eventualmente seguito da un moltiplicatore
- atomo =
  - RE tra parentesi tonde
  - Espressione []
  - ^ o $ o .
  - Singolo carattere

Espressioni regolari moderne (o estese)

Atomi speciali:

. indica qualsiasi carattere
^ indica l'inizio della linea
$ indica la fine della linea
[abc] indica qualsiasi carattere fra *a*, *b* o *c*
[a-z] indica qualsiasi carattere fra *a* e *z* compresi
[^dc] indica qualsiasi carattere che non sia né *d* né *c*

Moltiplicatori:
{n,m} indica da n a m occorrenze dell'atomo che lo precede
? indica zero o una occorrenza dell'atomo che lo precede
***** indica zero o più occorrenze dell'atomo che lo precede
+ indica una o più occorrenze dell'atomo che lo precede

Espressioni regolari - esempi

egrep '^att' miofile
ha come output tutte le righe di miofile che iniziano per **att**

egrep '^Nel.*vita\.$' miofile
ha come output tutte le righe di miofile che iniziano per **Nel** e finiscono per **vita.**

egrep '.es.e' miofile
righe che contengono 1 carattere seguito da **es** seguito da 1 carattere seguito da **e**

egrep '(day|night)*(shift|time)+' miofile

Grep – opzioni principali

Si possono alterare la modalità di ricerca di grep e l'aspetto dell'output per mezzo di opzioni da specificare sulla riga di comando prima della regexp e degli eventuali file.

-i rende l'espressione insensibile a maiuscole e minuscole
-v grep restituisce le linee che **non** contengono l'espressione

-l utile se passo a grep più file su cui cercare: restituisce solo i nomi dei file in cui l'espressione è stata trovata
-n restituisce anche il numero della riga contenente l'espressione

Grep - esercizi

1. trovare nella directory corrente la lista dei file scrivibili solo dall'utente proprietario e leggibili da chiunque
(hint: vedere l'output di ls -l)

2. elencare tutti gli utenti non appartenenti al gruppo root che hanno come shell di login bash
(hint: vedere il formato di /etc/passwd)

3. cercare nel dizionario di sistema le parole da 5 caratteri che iniziano per **c** e finiscono per **h**
(hint: vedere il file /usr/share/dict/words)

Estrazione dei campi – cut

Il comando **cut** permette di *tagliare* parti di righe. Su file organizzati a 'record' (linee) per i quali ogni record rappresenta una lista di 'campi' opportunamente delimitati, permette quindi di estrarre uno o più campi di ciascun record.

cut –c*elenco_caratteri* ritaglia le righe contando i caratteri

Es:
cut –c15 file restituisce solo il 15° carattere
cut –c8–30 file restituisce i caratteri dall'8° al 30°
cut –c–30 file restituisce i caratteri fino al trentesimo
cut –c8– file restituisce i caratteri dall'ottavo in poi

Estrazione dei campi – cut

cut –d*carattere_delimitatore* **–f***elenco_campi*
ritaglia le righe dove trova il delimitatore, generando una lista di campi

Es. se ci interessa estrarre solo il campo username dal file passwd:

Lo switch -s inibisce la riproduzione in output di righe che non contengono il delimitatore

cat /etc/passwd | cut –f1 –d: –s

Es. se nel campo note metto 'Nome Cognome' e voglio l'iniziale dei cognomi degli utenti:

cat /etc/passwd | cut –f5 –d: –s | cut –f2 –d' ' | cut –c1

Il comando rev

rev è un filtro che permette di invertire l'ordine dei caratteri di ogni linea dello stream in input verso lo stream di output.

L'utilità del comando è solitamente quella di accompagnare cut nella estrazione di campi la cui posizione sia nota relativamente al fine linea:

cat /etc/passwd | rev | cut -f1 -d: -s | rev

elabora il contenuto del file /etc/passwd nel sequente modo:
- inverte ogni linea
- prende il primo campo (→ l'ultimo campo dell'originale)
- inverte ogni linea (ripristina il campo selezionato)

I comandi sort e uniq

Per ordinare le linee di uno stream o per individuare le righe duplicate o uniche, unix mette a disposizione i filtri sort e uniq. Al solito, questi filtri operano sullo standard input se non c'è nessun argomento.

sort ordina alfabeticamente
sort –n ordina numericamente
sort –u elimina le entry multiple (equivale a sort | uniq)
sort –r inverte l'ordine

uniq elimina i duplicati
uniq –c indica anche il numero di occorrenze
uniq –d mostra solo le entry non singole

sort avanzato

sort è in grado di ordinare righe
- dando un significato numerico alle stringhe
- usando campi specifici anziché la riga intera, dall'inizio

sort -n ordina numericamente
sort -t. -k 1,1n -k 2,2n -k 3,3n -k 4,4n

Usando il punto come separatore di campo...

Ordina usando la stringa formata dai campi da 1 a 1, interpretata numericamente

Poi la stringa formata dal campo 2, poi il campo 3 ed infine il campo 4

… quindi ordina un elenco di IP address (a.b.c.d)

I comandi head e tail

head -N è un filtro che permette di estrarre solo le prime *N* righe di un file

tail -N è un filtro che permette di estrarre solo le ultime *N* righe di un file

Un'opzione particolare di **tail** è **–f** con cui, dopo aver mostrato le ultime righe di un file, lo si mantiene aperto e si visualizzano in tempo reale eventuali nuove righe che vi vengano appese da altri processi

Il comando wc

wc (word count) è un filtro di conteggio

wc -c conta i caratteri
wc -l conta le linee
wc -w conta le parole (stringhe separate da spazi)

Diff

Più frequentemente usato in modo interattivo, ma utile anche negli script, diff permette di mostrare le differenze tra due file.

```
riga uno
riga due
riga tre
riga quattro
riga cinque
riga sei
```

```
riga uno
riga tre
riga 4
riga cinque
riga 5bis
riga sei
```

```
$ diff filesinistro filedestro
2d1
< riga due
4c3
< riga quattro
---
> riga 4
5a5
> riga 5bis
```

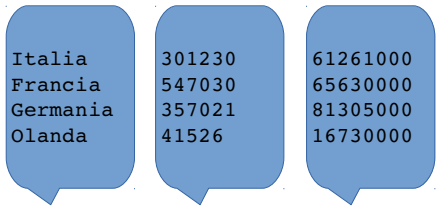
deleted

changed

added

Paste

In un certo senso l'opposto di cut: unisce "orizzontalmente" le righe di posizione omologa in vari file




```
$ paste nazioni superfici abitanti
Italia   301230   61261000
Francia  547030   65630000
Germania 357021   81305000
Olanda   41526 16730000
```

Join

Stesso principio di paste, ma anziché selezionare le righe sulla nase della posizione le unisce se iniziano con la stessa "chiave" (necessita di file ordinati in modo identico sulla chiave selezionata)


```
$ paste superfici abitanti
Italia   301230   61261000
Francia  547030   65630000
Germania 357021   81305000
Olanda   41526 16730000
```

Modifiche più complesse

Esistono altri comandi che consentono operazioni complesse sui file, come sed e awk. È riduttivo chiamarli filtri, dispongono di un vero e proprio linguaggio di programmazione per effettuare operazioni di manipolazione del testo. Vediamo solo qualche esempio pratico di utilità frequente:

sed **s/***vecchio_pattern***/***nuovo_valore***/** sostituisce in ogni riga un nuovo valore ad una parte di testo coincidente con un pattern

Es. inserisce la stringa "Linea:" all'inizio di ogni riga di passwd:
cat /etc/passwd | sed 's/^/Linea:/'
Abbrevia il titolo di dottore, accetta minuscolo o maiuscolo:
cat personale | sed 's/dottore/Dott./i'

Modifiche più complesse

awk è usato come evoluzione di cut, perché permette di considerare qualsiasi sequenza di caratteri come un unico delimitatore. Nell'uso più comune permette di superare uno dei più evidenti limiti di cut in presenza di più delimitatori consecutivi: ad esempio, **cut –f2 –d' '** se ci sono 2 spazi dopo il primo campo considera il 2° spazio come 2° campo.

Es. stampa il secondo campo del file, purchè sia separato dal primo da un numero qualunque di blanks
cat personale | awk '{print $2}'

Es. In un log che riporta il risultato di un'operazione come [varie…] stat=*esito* [varie…] estrae tutti gli esiti:
cat log | awk –F "stat=" '{print $2}' | awk '{print $1}'

Modifiche più complesse

awk è però dotato di un vero e proprio linguaggio di programmazione, del quale diamo solo qualche altro esempio.

1)A differenza di cut, non ha il concetto di "-f 5-" (dal quinto campo *in poi*) ma dispone in compenso di più potenti strumenti di manipolazione del testo:

```
cat file | awk '{print substr($0, index($0,$5)) }'
```

2)il codice è strutturabile in blocchi, e sono disponibili tutte le caratteristiche di un normale linguaggio di programmazione

```
ls -l | awk '{sum += $5} END {print sum}'
```

Modifiche più complesse – var string replace

La funzione di search e replace di sed è attuabile direttamente dalla shell sulle variabili:

```
mv "${FN}" "${FN/.bad/.bash}"
```

inside ${ … }	Action taken
name:number:number	Substring starting character, length
#name	Return the length of the string
name#pattern	Remove (shortest) front-anchored pattern
name##pattern	Remove (longest) front-anchored pattern
name%pattern	Remove (shortest) rear-anchored pattern
name%%pattern	Remove (longest) rear-anchored pattern
name/pattern/string	Replace first occurrence
name//pattern/string	Replace all occurrences

Modifiche più complesse – tr

Per sostituire più rapidamente singoli caratteri (senza regex), si può utilizzare **tr**:

```
tr 'A-Z' 'a-z'
```
trasforma ordinatamente le maiuscole in minuscole

```
tr ';:.!?' ','
```
sostituisce qualsiasi occorrenza dei caratteri nel primo set con ,

```
tr ';:.!?' ',-'
```
in generale, se il secondo set è più limitato del primo set, il suo ultimo carattere viene ripetuto quanto basta a generare la corrispondenza 1:1. In questo caso quindi ; → , : → - . → - ! → - ? → -

```
tr -d '\r'
```
elimina ogni occorrenza del carriage return

Cstruzione di linee di comando complesse con xargs

xargs <comando> si aspetta sullo standard input un elenco di stringhe, ed invoca poi **comando** con tali stringhe come argomenti. L'uso di xargs permette il passaggio di un numero qualsiasi di argomenti (se si eccede la lunghezza massima della command line, xargs divide in più invocazioni e raggruppa le invocazioni in modo da ridurre il carico del sistema.)

Es. alternativamente a **–exec** dell'esempio di find:
find /usr/src -name '*.c' -size +100k -print | xargs cat

Costruzione di linee di comando complesse con xargs

Es. manda un messaggio di posta vuoto con oggetto “guasto! uscite” a tutti gli utenti:

cat /etc/passwd | cut –f1 –d: | xargs mail –s ‘guasto! uscite’

Nota: si poteva anche con i backtick:

mail –s ‘guasto! uscite’ `cat /etc/passwd | cut –f1 –d:`

xargs ha però molte opzioni che lo rendono più flessibile:

- sostituzione di parti degli argomenti e definizione dei delimitatori di argomento
- esecuzione interattiva con conferma di ogni linea di comando generata
- definizione dei limiti di numero di argomenti e di lunghezza delle command line

Esercizi di costruzione pipeline con filtri

1 - Contare quanti file esistono con una certa estensione, definita come la stringa posta dopo l'ultimo carattere "punto" presente nel nome del file, per tutte le estensioni trovate nei file presenti nel direttorio corrente e sottodirettori. Limitare l'output alle sole 5 estensioni più numerose.

2 - Utilizzare le funzionalità avanzate di sort (switch -k e -t) per ordinare per data di nascita, a parità di data per cognome, a parità di entrambi per nome un file nel formato
Nome/Cognome/GG/MM/AAAA
oppure
Nome/Cognome/AAAA/MM/GG

Ricerca di file

Il filesystem di unix/Linux, specialmente in installazioni di tipo server, ospita comunemente centinaia di migliaia di file. Cercare un file o una directory all'interno di un insieme così vasto di file può essere una operazione frustrante, nonostante l’agevolazione costituita dall'ordine gerarchico del filesystem e dalle linee guida standard per la collocazione delle risorse (Es. se stiamo cercando un file di configurazione, con ogni probabilità questo si trova sotto /etc)

Sono per questo a disposizione due utility di ricerca molto potenti, una ottimizzata per la flessibilità, l’altra per la velocità.

Ricerca di file - find

find ricerca in tempo reale i file che soddisfano una combinazione di criteri, ad esempio file che:

- abbiano un nome che contenga una espressione data
- siano stati usati in un periodo specificato
- abbiano dimensione compresa tra un minimo e un massimo
- siano di un tipo specificato (file, dir, link simbolici, ...)
- siano di proprietà di un utente o di un gruppo specificati
- abbiano certi permessi di accesso
- si trovino ad una certa profondità dell'albero del filesystem
- soddisfino una o più delle precedenti condizioni

Archiviazione di file

Per poter agevolmente memorizzare e trasferire una molteplicità di file, eventualmente senza perdere le proprietà associate a ciascuno (ownership, permessi, timestamps…) è comune avvalersi di *tar*. La sintassi prevede che debba essere specificato esattamente uno dei seguenti comandi:

-A	concatena più archivi
-c	crea un nuovo archivio
-d	trova le differenze tra archivio e filesystem
-r	aggiunge files ad un archivio
-t	elenca il contenuto di un archivio
-u	aggiorna files in un archivio
-x	estrai files da un archivio
--delete	cancella files da un archivio

Archiviazione di file

Le origini di tar risalgono ai tempi dei nastri magnetici (il nome stesso è acronimo di Tape ARchiver) quindi di default assume che l'archivio sia su /dev/tape. L'uso "moderno" prevede però sempre la presenza dell'opzione "-f *filename*" per specificare un file di archiviazione. Filename può essere "-" per indicare lo standard input o lo standard output.
Le altre opzioni comunemente usate sono:

-p	conserva tutte le informazioni di protezione (funziona solo per root, un utente standard quando ricrea i file estraendoli da un archivio è forzato a dargli la sua ownership e a crearli in accordo con la sua umask)
-v	stampa i dettagli durante l'esecuzione
-z/-j	(de)comprime con gzip/bzip2

Archiviazione di file

Tra le numerose opzioni meno comuni, ma all'occorrenza molto utili (vedere la man page) citiamo le possibilità:

- di leggere da un file l'elenco dei file da includere o da escludere dall'archiviazione,
- di decidere come gestire i symlink,
- di decidere le modalità di sovrascrittura o cancellazione dei file esistenti, …

Esempi:
creazione tar cvpzf users.tgz /home/*
(tutti i file sono inseriti come "home/….")

estrazione cd /restore ; tar xvpzf users.tgz
(ricrea la gerarchia "home/…." nella dir. /restore)

Compressione di file

I compressori standard di Unix operano su un singolo file, quindi la funzionalità tipica di WinZip, o RAR, o simili, si ottiene con l'accoppiata tar+compressore.

Il compressore più comune è *gzip*:
gzip *file* → comprime *file* e genera *file.gz*
gunzip *file.gz* → decomprime *file.gz* e genera *file*

Con sintassi analoga si possono usare i più moderni *bzip2*/*bunzip2*

- L'opzione "-c" permette l'uso dei compressori come filtri che scrivono su stdout
- gunzip/bunzip2 equivalgono a gzip/bzip2 **-d**

Compressione di file

I pacchetti gzip e bzip2 di solito forniscono degli alias per invocare i comandi più comuni direttamente su file compressi:

zcat *file.gz* == gzip -dc *file.gz*
zgrep *file.gz* == gzip -dc *file.gz* | grep
zless *file.gz* == gzip -dc *file.gz* | less

(idem per bzcat, bzgrep, bzless)

L'opzione z di tar può essere interpretata analogamente:

tar czf *arch.tgz* * == tar cf - * | gzip -c > *arch.tgz*
tar xzf *arch.tgz* == gzip -dc *arch.tgz* | tar xf -

Uso di find

il programma find si utilizza con la seguente convenzione:

find [FILE...] [EXPRESSION]

dove FILE rappresenta il o i percorsi di ricerca e EXPRESSION indica l'elenco delle caratteristiche richieste o delle operazioni desiderate.

Es. per ricercare sotto /usr/src tutti i file che finiscono per '.c', hanno dimensione maggiore di 100K, ed elencarli sullo standard output:

find /usr/src -name '*.c' -size +100k -print

Uso di find

Una delle opzioni più potenti di find permette, per ciascun oggetto individuato secondo i criteri impostati, di invocare l'esecuzione di un comando:

Es. mostra il contenuto dei file trovati

find /usr/src -name '*.c' -size +100k **-exec** cat **{} \;**

il comando che segue **–exec** viene lanciato per ogni file trovato, la sequenza **{}** viene sostituita di volta in volta con il nome del file, **\;** è necessario per indicare a find la fine del comando da eseguire.

Es. elenca solo i file che contengono *stringadacercare*

find / -type f **-exec** grep -l stringadacercare **{} \;**

Ricerca di file - locate

find è molto potente e flessibile, ma lanciato su filesystem grandi è molto lento poiché percorre le directory una per una.

locate ricerca file che hanno un certo pattern nel nome senza accedere direttamente a tutte le directory del filesystem coinvolte ma utilizzando un database di file che opera da 'cache'. È quindi molto più veloce di find, ma "fotografa" la situazione all'istante in cui il database è viene aggiornato (mediante l'utility **updatedb**) e non permette di filtrare la ricerca con criteri flessibili come quelli di find.