

I filtri

Un **filtro** è un comando che manipola la sua standard input e invia il risultato sulla standard output.

Un filtro comune è `sort`. Usiamolo per leggere dalla tastiera:

```
/home/user1> sort
Marco 347898
Antonio 050 76564
```

Quando avete finito di inserire nomi, battete **Ctrl D**: e `sort` ordinerà alfabeticamente le righe di input:

```
^D
Antonio 050 76564
Marco 347898
```

Ovviamente, salvo che per un breve test, è più comune ordinare i dati memorizzati in un file e salvare i risultati su un altro file, p. es.

```
/home/user1> sort < telefoni > telefoni.ord
```

Esercizio: create un file `telefoni` di almeno 40 numeri di telefono, nella forma mostrata sopra, e ordinatela come detto.

`sort` viene usato spesso in pipe con altri comandi, P. es. per ordinare gli utenti presenti sul sistema

```
/home/user1> who | sort
/home/user1> who | sort | lpr
/home/user1> who | sort | tee users.save | lpr
```

Secondo il solito meccanismo, `sort` ammette una lista di input file, inoltre può mandare output su un file con l'opzione `-o`:

```
/home/user1> sort -otelefoloni telef.1 telef.2
```

ciò equivale a (qui `sort` comunica solo con standard I/O):

```
/home/user1> (cat telef.1 telef.2 | sort) > telefoni
```

Opzioni di sort

`sort -r` rovescia l'ordine

`sort -n` ordina numericamente (e tiene conto di `-e` . decimale):

```
/home/user1> echo '-80
> -1
> 19
> 123' > numeri
/home/user1> sort < numeri
-1
-80
123
19
/home/user1> sort -n < numeri
-80
-1
19
123
```

`sort -u` elimina i duplicati:

```
/home/user1> sort -u
aa
aa
bb
^D
aa
bb
/home/user1> sort
aa
aa
bb
^D
aa
aa
bb
```

Altre opzioni di sort

Gli spazi hanno codice ascii 32 e le maiuscole hanno un codice più basso di tutti quelli delle minuscole.

Per provare, usate `od -hc` che legge un file o la standard input, mostrandone ogni byte come *hexadecimal* e *char*:

```
/home/user1> od -hc
a A
000000000 61 20 41 0d 0a
^D
000000005
```

`sort -n` (ordina numericamente) e ignora spazi e tab iniziali:

```
/home/user1> sort
b
a
^D
b
a
/home/user1> sort -n
b
a
^D
a
b
```

`sort -f` ignora la differenza tra maiuscole e minuscole

```
/home/user1> sort
a
B
^D
B
a
/home/user1> sort -f
a
B
^D
a
B
```

Altre opzioni di sort

`sort -d` ignora i segni di interpunzione, come un dizionario:

```
/home/user1> sort -d
D'Ernesto
DAVIDE
^D
DAVIDE
D'Ernesto
/home/user1> sort
D'Ernesto
DAVIDE
^D
D'Ernesto
DAVIDE
```

`sort -m f1 f2 ...` fonde in maniera ordinata i file *f1 f2 ...* ma non li ordina all'interno (è più veloce di `sort f1 f2`, ma il risultato è ordinato solo se *f1* e *f2* sono già ordinati):

```
$ echo 'b
> a' > f1
$ echo 'z
> v' > f2
$ cat f1 f2
b
a
z
v
$ sort -m f2 f1
b
a
z
v
$ sort f2 f1
a
b
v
z
```

Ordinamento a chiave o per campi

L'informazione contenuta in un file testo è suddivisa in righe. Spesso anche le righe sono suddivise in **campi**.

sort riconosce i campi in quanto separati da spazi:

```
7689 Benso Camillo TO 011 1860-61
9854 Verdi Giuseppe, via Rossini, Parma
3687 Vialli Gian Luca via Dalla Samp GE 010 878787
```

In quest'esempio, che mostra alcune righe del file agenda:

- le righe 1 e 2 contengono 6 campi, la 3 contiene 10 campi.
 - i campi 1 (num d'ordine) e 2 (cognome) hanno lo stesso significato in ogni riga, ma quello degli altri campi varia.
- sort ordina tenendo conto di tutta la riga e non dei campi:

```
/home/user1> sort agenda
3687 Vialli Gian Luca via Dalla Samp GE 010 878787
7689 Benso Camillo TO 011 1860-61
9854 Verdi Giuseppe, via Rossini, Parma
```

Invece `sort +m -n` ordina su una **chiave** cosiffatta:

- l'**inizio** della chiave si determina saltando **m** campi
- la **fine** della chiave si determina saltando **n** campi

Dunque l'ordinamento sul campo 2 (cognome) si fa con:

```
/home/user1> sort +1 -2 agenda
9854 Verdi Giuseppe, via Rossini, Parma
3687 Vialli Gian Luca via Dalla Samp GE 010 878787
7689 Benso Camillo TO 011 1860-61
```

Gli spazi che precedono Verdi sono parte del campo.

Per evitare che Verdi sia il primo: opzione `-b`

```
/home/user1> sort -b +1 -2 agenda
7689 Benso Camillo TO 011 1860-61
9854 Verdi Giuseppe, via Rossini, Parma
3687 Vialli Gian Luca via Dalla Samp GE 010 878787
```

Ordinamento a chiave (cont.)

Se un file contiene campi in numero e di natura variabile, usare spazi per separare i campi crea problemi:

```
7689 Benso Camillo TO 011 1860-61
9854 Verdi Giuseppe, via Rossini, Parma
3687 Vialli Gian Luca via Dalla Samp GE 010 878787
```

P.es. il 3° o il 5° campo contengono informazione eterogenea, quindi usare la via o la città come chiave è impossibile.

Quindi è meglio usare p.es. virgole come separatori di campo.

P. es. nel file agenda.l:

```
/home/user1> cat agenda.l
9854,Verdi ,Giuseppe ,via Rossini ,Parma,
3687,Vialli,Gian Luca,via Dalla Samp,GE ,010 878787
3688,Vialli,Gian Luca,via Cerea ,TO ,011 777777
7689,Benso ,Camillo ,TO ,011 1860-61
```

“`sort -t'c'` ordina usando **c** come separatore di campi.

```
/home/user1> sort -t, +1 -2 agenda.l
7689,Benso ,Camillo ,TO ,011 1860-61
9854,Verdi ,Giuseppe ,via Rossini ,Parma,
3687,Vialli,Gian Luca,via Dalla Samp,GE ,010 878787
3688,Vialli,Gian Luca,via Cerea ,TO ,011 777777
```

Si può anche ordinare su più campi contemporaneamente.

P.es. per ordinare sul nome e, a parità di nome, sulla città:

```
/home/user1> sort -t, +1 -2 +4 -5 agenda.l
7689,Benso ,Camillo ,TO ,011 1860-61
9854,Verdi ,Giuseppe ,via Rossini ,Parma,
3688,Vialli,Gian Luca,via Cerea ,TO ,011 777777
3687,Vialli,Gian Luca,via Dalla Samp,GE ,010 878787
```

Per ogni campo preceduto da `-b` si ignorano gli spazi (es. `' TO'`):

```
/home/user1> sort -t, +1 -2 -b +4 -5 agenda.l
7689,Benso ,Camillo ,TO ,011 1860-61
9854,Verdi ,Giuseppe ,via Rossini ,Parma,
3687,Vialli,Gian Luca,via Dalla Samp,GE ,010 878787
3688,Vialli,Gian Luca,via Cerea ,TO ,011 777777
```

Ordinamento a chiave (cont.)

Si può raffinare una specifica di campo usando il punto decimale: la chiave di sort +5.4 inizia dopo 5 campi e 4 caratteri (del 6°).

```
/home/user1> cat agenda1
9854,Verdi ,Giuseppe ,via Rossini ,Parma ,
3687,Vialli,Gian Luca,via Dalla Samp,GE ,010 878787
3688,Vialli,Gian Luca,via Cerea ,TO ,011 777777
7689,Benso ,Camillo ,TO ,011 1860-61
/home/user1> sort -t, +5 -6 agenda1 ordina su num. di tel.
9854,Verdi ,Giuseppe ,via Rossini ,Parma ,010 878787
3687,Vialli,Gian Luca,via Dalla Samp,GE ,011 1860-61
7689,Benso ,Camillo ,TO ,011 777777
3688,Vialli,Gian Luca,via Cerea ,TO ,011 777777
/home/user1> sort -t, +5.4 -6 agenda1 ignora il prefisso
9854,Verdi ,Giuseppe ,via Rossini ,Parma ,
7689,Benso ,Camillo ,TO ,011 1860-61
3688,Vialli,Gian Luca,via Cerea ,TO ,011 777777
3687,Vialli,Gian Luca,via Dalla Samp,GE ,010 878787
```

La chiave di sort -5.3 finisce dopo 5 campi e 3 caratteri (del 6°) (confrontare il seguente esempio con sort +5 -6 più sopra):

```
/home/user1> sort -t, +5 -5.3 agenda1
9854,Verdi ,Giuseppe ,via Rossini ,Parma ,
3687,Vialli,Gian Luca,via Dalla Samp,GE ,010 878787
3688,Vialli,Gian Luca,via Cerea ,TO ,011 777777
7689,Benso ,Camillo ,TO ,011 1860-61
```

Attenzione: certe implementazioni di sort interpretano le

specifiche di chiave sort +m -n diversamente:

vengono considerati i campi da m fino a n-1.

Esercizio: creare un'agenda di 100 nomi con campi come sopra:

identificatore, cognome, nome, indirizzo, città, telefono

e ordinare:

- per cognome e, insieme, per città, su un file agenda.1
- per città e, insieme, per nome, trascurando la differenza tra maiuscole e minuscole, su un file agenda.2

uniq

uniq f g elimina le righe **adiacenti** ripetute dal file f e salva il risultato in g. Se g manca, uniq scrive sulla standard output.

```
/home/user1> cat nomi
Tiberio Gracco
Tiberio Gracco
Cao Gracco
Tiberio Gracco
/home/user1> uniq nomi
Tiberio Gracco
Cao Gracco
Tiberio Gracco
```

L'opzione -d dà solo le righe ripetute, -u elimina ogni ripetizione e -c conta le righe ripetute adiacenti:

```
/home/user1> uniq -d nomi
Tiberio Gracco
/home/user1> uniq -u nomi
Cao Gracco
Tiberio Gracco
/home/user1> uniq -c nomi
 2 Tiberio Gracco
 1 Cao Gracco
 1 Tiberio Gracco
```

uniq -n trascura i primi n campi nel confrontare le righe

```
/home/user1> uniq -1 nomi
Tiberio Gracco
```

comm e diff

comm seleziona le righe comuni a due file ordinati:

```
/home/user1> cat fornit.90
fiat
ibm
olivetti
/home/user1> cat fornit.91
fiat
nissan
olivetti
/home/user1> comm fornit.90 fornit.91
fiat
nissan
olivetti
solo      nissan
file1     solo
file2     righe
          comuni
```

Le opzioni -1 -2 -3 sopprimono la colonna corrispondente.

Il comando diff f1 f2 trova le differenze tra f1 e f2:

```
/home/user1> diff fornit.90 fornit.91
2c2
< ibm
---
> nissan
```

NB: < precede i riferimenti a f1, > quelli a f2, e:

- c indica le righe da cambiare,
 - d le righe da cancellare,
 - a le righe da aggiungere
- per rendere i file f1 uguale a f2.

Esercizio: creare un file di fornitori fornit.90, quindi eliminare i fornitori stranieri, sostituendoli, in parte, con italiani. Applicare comm e diff ai due file.

tr

tr stringa1 stringa2 copia la standard input sulla standard output, traducendo i caratteri in stringa1 in quelli in stringa2:

```
/etc> tr a-e 12
abcdef^D
12222f
/etc> tr a-z A-Z
abcdef^D
ABCDEF
```

tr -c traduce i caratteri **non in** stringa1 in quelli in stringa2:

```
/etc> tr -c a-z +
casa, dolce casa^D
casa++dolce+casa+
```

tr -s elimina i caratteri tradotti duplicati:

```
/etc> tr -cs a-z +
casa, dolce casa^D
casa+dolce+casa+
```

Il seguente comando cerca le 10 parole più comuni nel file f

```
$ (tr -sc A-Za-z '\012' | traduci ogni non-lettera in newline (10)
sort | ordina
uniq -c | elimina duplicati e conta occorrenze
sort -n | ordina secondo contatore occorrenze
tail) < f prendi ultime 10 righe
```

Si noti come la pipe | consenta di continuare l'input a capo, e come le parentesi tonde siano necessarie.

Esercizio: eliminare le parentesi tonde nell'esempio precedente. (suggerimento: si usi cat).

Altri filtri: `fmt`, `fold`, `n1`

`fmt -l n` elimina gli 'a capo' dal suo input e ne inserisce di nuovi, per produrre in standard output righe di lunghezza quanto più vicina per difetto a `n`.

`fold -n` spezza le righe di input ogni `n` caratteri, ma non modifica gli 'a capo' preesistenti.

```
$ cat leap
Sempre caro mi fu quest'ermo colle
e questa siepe che da tanta parte
$ fmt -l 15 leap
Sempre caro mi
fu quest'ermo
colle e questa
siepe che da
tanta parte
$ fold -15 < leap
Sempre caro mi
fu quest'ermo c
olle
e questa siepe
che da tanta pa
rte
```

`n1` legge la standard input e numera le righe:

```
/home/user1> cat leap
Sempre caro mi fu quest'ermo colle
e questa siepe che da tanta parte
/home/user1> n1 leap
1 Sempre caro mi fu quest'ermo colle
2 e questa siepe che da tanta parte
```

`grep`

Una **stringa** è una sequenza di caratteri.

Un **pattern** è una notazione per specificare un insieme di stringhe. `grep pattern file` cerca una stringa che si accoppi (**match**) con **pattern** in **file** e stampa le righe in cui la ricerca ha avuto successo. `grep pattern` usa la standard input per default, p. es.:

```
/home/user1> ls | grep affari
affari
```

Per specificare pattern, la notazione è simile a quella della shell, anzi spesso il pattern di `grep` va protetto dall'interpretazione della shell, racchiudendolo tra apici.

<code>c</code>	il carattere <code>c</code> sta per se stesso (se non è un metacarattere)
<code>\c</code>	<code>lo \</code> elimina ogni significato speciale di <code>c</code>
<code>^ e \$</code>	inizio e fine riga
<code>.</code>	ogni singolo carattere
<code>[...]</code>	uno qualsiasi dei caratteri ...; si possono usare range come <code>a-z</code>
<code>[^...]</code>	uno qualsiasi dei caratteri non in ...; si possono usare range
<code>p*</code>	zero o più occorrenze del pattern <code>p</code>
<code>p1 p2</code>	pattern <code>p1</code> seguito da <code>p2</code>

Esempi: `[^0-9]` è ogni carattere non numerico,

`[[-]]` è: `[o - o]` (`[-]` non sono metacaratteri se racchiusi tra `[]`)

`ls -l | grep '^d'` mostra le subdirectory (righe inizianti per `d`)

`[a-zA-Z]*` è la stringa vuota o una qualsiasi stringa alfabetica

`*` sta per ogni stringa (`*` va preceduto da un pattern, da solo è un errore)

`*x` è ogni stringa che finisce per `x`

`xy*` non è `xyxyxy... ma xyxy... perché *` lega più forte della concaten.

`grep '^[^:]*::' /etc/passwd` cerca gli utenti senza password, infatti le righe di `/etc/passwd` hanno la forma:

`nomeutente:passwordcodificata:....`

e il pattern è: inizio riga, 0 o più caratteri diversi da `:`, seguiti da `:`

egrep

egrep è una versione estesa di grep, i pattern si formano come quelli di grep e in più con:

- $p+$ una o più occorrenze del pattern p
- $p?$ zero o una sola occorrenza del pattern p
- $p1/p2$ pattern $p1$ oppure pattern $p2$
- (p) pattern p (le parentesi servono per raggruppare più pattern)

Questo consente di specificare più patterns che con grep, p. es.:

$(xy)^*$ sta per: la stringa vuota, xy , $xyxy$, $xyxyxy$, ...

chiesa | chiave e chi (esa | ave) stanno entrambi per chiesa o chiave

egrep -f pat può cercare i pattern specificati in un file pat, pat può contenere pattern multipli, ognuno su una riga.

NB: quando il pattern è sulla riga di comando è spesso tra ' e ', per evitare che la shell interpreti i metacaratteri del pattern; nei pattern su file ciò non è necessario:

```
/etc> cat aeiou
^[^aeiou]*a[^aeiou]*e[^aeiou]*i[^aeiou]*o[^aeiou]*u[^aeiou]*$
```

Il pattern del file aeiou è: inizio riga (^), una sequenza di 0 o più non vocali, una a, una sequenza di 0 o più non vocali, una e, ... in altre parole: una riga contenente solo le vocali aeiou nell'ordine

Cerchiamo il pattern di aeiou nel dizionario /etc/words.

NB: il comando c incolonna l'output, ma non ogni UNIX ce l'ha:

```
/etc> egrep -f aeiou words | c
/etc> egrep -f aeiou words | c
abstemious      abstemiously  arsenious      caesious
facetious       facetiously
```

Sul vostro sistema dovrebbe esserci pure un dizionario, spesso è in /usr/dict/words o nella dir /usr/lib

egrep - fgrep

Per cercare parole con le lettere in ordine e con almeno 6 lettere:

```
/etc> cat ord
^a?b?c?d?e?f?g?h?i?j?k?l?m?n?o?p?q?r?s?t?u?v?w?x?y?z?.$
/etc> egrep -f ord words | grep '.....' | c
abhors almost begins bijoux biopsy chinos chintz
```

fgrep è una versione veloce (fast) di grep che cerca solo stringhe semplici, non pattern complessi

Leggendo i pattern da un file, fgrep ne può cercare diversi “in parallelo”, più velocemente.

Principali opzioni per la famiglia grep:

- n dà i numeri delle righe in cui il *pattern* è stato trovato
- c conta solo le righe che contengono il pattern
- v mostra tutte le righe che **non** contengono il pattern
- i ignora case: accoppia minuscole in *pattern* anche a maiuscole
- h non visualizza i nomi dei file in una ricerca su più file.
P. es., se non si usa -h:

```
/home/user1> cat f1
Sempre caro mi fu quest'ermo colle
/home/user1> cat f2
e questa siepe che da tanta parte
/home/user1> egrep 'r[mt]' f1 f2
f1: Sempre caro mi fu quest'ermo colle
f2: e questa siepe che da tanta parte
```

Esercizio su grep

Dato un file agenda:

- visualizzare le righe contenenti m
- visualizzare le righe contenenti s o S
- visualizzare le righe che non contengono a o e
- contare le righe che contengono la stringa and
- in /etc/passwd, trovare gli utenti che non usano /bin/sh
- scelte 4 parole, si trovi un pattern comune ad esse.
- spiegare la differenza tra fgrep ed egrep

Dire cosa individuano i seguenti:

Pattern	Risposte
^\$	riga vuota
.	ogni riga non vuota
^	ogni riga
^thing	thing a inizio di riga
thing\$	thing in fine di riga
^thing\$	riga che contiene solo thing
thing.\$	thing seguito da un carattere qualsiasi a fine riga
thing\.\$	thing seguito da un punto a fine riga
[t]hing	
thing[0-9]	
thing[^0-9]	
thing[0-9][^0-9]	thing1 seguito da qualsiasi stringa e da thing2
thing1.*thing2	come sopra, ma è tutta la riga
^thing1.*thing2\$	

sed: uno stream editor

La forma di base di sed è:

```
$ sed 'lista di comandi d'ed' f1 f2 ...
```

sed legge le righe f1 f2 ... e applica a ognuna i comandi di ed.

sed -f *cmdfile* prende i comandi da *cmdfile*

Esempi (NB: sotto → sta per il tasto Tab):

```
sed 's/UNIX/UNIX(TM)/g'
```

gli ' ' servono a quotare i metacaratteri di shell

```
/home/user1> du -a ch*
1 ch.1
1 ch.2
/home/user1> du -a ch* | sed 's/.*→//'
```

sed 's/ .* / / ' sostituisce tutto ciò che sta tra il primo e l'ultimo spazio bianco con un solo spazio bianco.

sed 's/^ /→/ ' rientra le righe di un tab

sed ' /./s/^ /→/ ' fa lo stesso; inoltre il pattern precedente s restringe le righe considerate a quelle che contengono almeno un carattere (.)

sed ' /^\$/!s/^ /→/ ' fa lo stesso, perché ! nega il pattern che lo precede e / ^\$ / prende le righe vuote

sed -n elimina la stampa per default di tutte le righe.

La stampa esplicita si fa con p, preceduto da un **pattern**

selettore:

```
/home/user1> sed -n /a/p
coso
casa
casa
```

sed (cont.)

Per convertire ogni sequenza di almeno uno spazio bianco o tab in un ritorno a capo, si deve andare a capo sulla riga di comando:

```
/home/user1> sed 's/[ ]\+ →]*/\n
> /g'
casa dolce      casa
casa
dolce
casa
```

Il comando `d` cancella: `sed '/pattern/d'` stampa tutto, ma cancella ogni riga contenente *pattern*.

Il comando `q` esce: `sed '/pattern/q'` stampa finché trova una riga contenente *pattern*, poi esce.

I pattern selettori si possono usare a coppie, come in `ed`.

P. es.:

```
sed -n '20,30p' stampa le righe da 20 a 30
sed '1,10d' stampa cancellando le righe da 1 a 10
sed '1,/^$/d' stampa cancellando le righe dalla 1 alla
prima vuota (tutte se nessuna è vuota)
sed $d cancella la riga $ cioè l'ultima
(NB $ ha due usi: fine riga in un pattern,
ultima riga in un contatore di riga).
```