Comandi Unix

G. Lettieri

May 24, 2005

Introduzione

Negli esempi seguenti, si suppone che la directory corrente contenga i file:

```
fileA
fileB
fileC
fileD
```

Il cui contenuto è il seguente:

`fileA`

```
aa
bbbb
```

`fileB`

```
ccbbb acc
dddcaa  aaaccab
```

`fileC`

```
aaa
bbb
bbb
bbb
aaa
ccc
```

`fileD`

```
a1      a2      a3      a4      a5
b1      b2      b3      b4      b5
c1      c2      c3      c4      c5
```

Comandi

cat

Concatena file e invia sull'uscita standard.

`cat` *file1 file2* ...

Apre in sequenza *file1*, *file2*, etc. e ne invia il contenuto all'uscita standard. Se (e solo se) nessun nome di file viene specificato, il comando invia all'uscita standard ciò che riceve sull'ingresso standard.
Esempi:

```
$ cat fileA fileB
aa
bbbb
ccbbb acc
dddcaa  aaaccab
$ cat fileB
ccbbb acc
dddcaa  aaaccab
```

cc

Compilatore C.

`cc` [`-o` eseguibile] *sorgente1 sorgente2* ...

Compila tutti i file sorgente (che possono contenere programmi scritti in C o in Assembler), collega il risultato con le librerie di sistema e ottiene un nuovo file eseguibile (binario). Il nome del file eseguibile sarà `a.out` (nome storico che vuol dire Assembler OUTput), oppure il nome *eseguibile* specificato tramite l'opzione `-o`, se presente.
Esempi:

```
$ echo 'int main() { printf("Hello, world!\n"); return 0; }' > hello.c
$ cc -o hello hello.c
$ ls -l hello
-rwxr-xr-x  1 giuseppe users 7351 May 24 14:49 hello
$ ./hello
Hello, world!
```

chmod

Cambia i permessi di accesso di uno o più file o directory.

`chmod` *permessi file1 file2* ...

I permessi dei file (o directory) specificati verranno cambiati in accordo al parametro *permessi*, che deve essere una lista di uno o più elementi separati da virgole (senza spazi nel mezzo), dove ogni elemento è così composto: zero o più tra le lettere `u` (proprietario), `g` (gruppo), `o` (altri); uno tra i caratteri `+` (aggiungi il permesso), `-` (togli il permesso), `=` (rendi i permessi esattamente

uguali a); zero o più tra le lettere `r` (permesso di lettura), `w` (permesso di scrittura), `x` (permesso di esecuzione), `s` (set user/group id bit), `t` (per le directory: permetti la cancellazione solo ai proprietari dei file).
Esempi:

```
$ ls -l fileA
-rw-r--r--  1 giuseppe users 8 May 23 12:02 fileA
$ chmod ug=rw,o= fileA
$ ls -l fileA
-rw-rw----  1 giuseppe users 8 May 23 12:02 fileA
$ chmod -r fileA
$ ls -l fileA
--w--w----  1 giuseppe users 8 May 23 12:02 fileA
$ mkdir dir1
$ chmod +t dir1
$ ls -l | grep dir1
drwxr-xr-t  2 giuseppe users 1024 May 23 19:24 dir1
```

chown

Cambia il proprietario (e il gruppo) di uno o più file.

`chown` *utente*[`:`*gruppo*] *file1 file2* ...

cp

Copia uno o più file.

`cp` *file1 file2*

`cp` *file1 file2* ... *directory*

Nella prima forma, copia il contenuto del file *file1* nel file *file2* (creandolo se non esiste, e cancellando il vecchio contenuto di *file2*, se esisteva già).

Nella seconda forma, copia tutti i file nella directory *directory* (che deve esistere), ognuno col proprio nome (e cancellando eventuali file con lo stesso nome già presenti nella directory *directory*).
Esempi:

```
$ cp fileA nuovo
$ ls -il fileA nuovo
11 -rw-r--r--  1 giuseppe users 8 May 23 12:02 fileA
15 -rw-r--r--  1 giuseppe users 8 May 23 19:24 nuovo
$ mkdir dir1
$ cp file* dir1
$ ls -l dir1
total 4
-rw-r--r--  1 giuseppe users  8 May 23 19:24 fileA
-rw-r--r--  1 giuseppe users 26 May 23 19:24 fileB
-rw-r--r--  1 giuseppe users 24 May 23 19:24 fileC
-rw-r--r--  1 giuseppe users 45 May 23 19:24 fileD
```

cut

Rimuove sezioni da ogni linea di uno o più file.

`cut` [`-d`*delimitatore*] `-f`*campi file1 file2* ...

Suddivide ogni linea di ingresso in campi separati dal carattere TAB (oppure dal carattere *delimitatore*, se specificato con l'opzione `-d`). Quindi, di ogni linea, stampa solo i campi richiesti tramite il parametro `-f`*campi*. I campi sono numerati a partire da 1. L'argomento *campi* deve essere una lista di uno o più intervalli, separati da virgola, dove ogni intervallo è un singolo numero, oppure due numeri separati dal segno "`-`".
Esempi:

```
$ cut -f2 fileD
a2
b2
c2
$ cut -f1,3 fileD
a1      a3
b1      b3
c1      c3
$ cut -f1-3,5 fileD
a1      a2      a3      a5
b1      b2      b3      b5
c1      c2      c3      c5
$ cut -f2- fileD
a2      a3      a4      a5
b2      b3      b4      b5
c2      c3      c4      c5
```

Si noti che, se in un intervallo manca il secondo numero, si intende che l'intervallo si estende fino all'ultimo campo della linea.

```
$ ls -l | tr -s ' ' | cut -f1,9 -d' '
total
-rw-r--r-- fileA
-rw-r--r-- fileB
-rw-r--r-- fileC
-rw-r--r-- fileD
```

Nell'ultimo esempio, viene usato il carattere "spazio" come delimitatore (per visualizzare il primo e il nono campo dell'uscita di `ls -l`). Si noti la necessità di proteggere lo spazio con gli apici (altrimenti, la shell lo avrebbe eliminato) e di trasformare le sequenze di spazi in un unico spazio (altrimenti, sequenze di spazi nell'uscita di `ls -l` avrebbero contato come più campi).

date

Mostra la data e l'ora di sistema.

`date`

Esempi:

```
$ date
Mon May 23 19:24:52 CEST 2005
```

diff

Trova le differenze tra due file.

`diff` *file1 file2*

Mostra sull'uscita standard le linee diverse tra file1 e file2 (non mostra niente se i due file contengono esattemente le stesse linee). Restituisce 0 se i due file differiscono.
Esempi:

```
$ cp fileA nuovo
$ diff fileA nuovo
$ sed s/aa/zz/ fileA > nuovo2
$ cat nuovo2
zz
bbbb
$ diff fileA nuovo2
1c1
< aa
---
> zz
$ if diff fileA nuovo >/dev/null; then echo "Diversi!"; else echo "Uguali!"; fi
Diversi!
$ if diff fileA nuovo2 >/dev/null; then echo "Diversi!"; else echo "Uguali!"; fi
Uguali!
```

echo

Mostra una linea di testo.

`echo` *stringa1 stringa2 ...*

Copia sull'uscita standard tutte le stringhe ricevute come argomento, separate da uno spazio l'una dall'altra, e infine va a nuova linea. Invocato senza argomenti, invia sull'uscita standard un solo "a capo".
Esempi:

```
$ echo Hello      World
Hello World
$ echo "Hello      World"
Hello      World
$ echo

$ echo $HOME
/home/giuseppe
```

expr

Valuta espressioni aritmetiche.

`expr` *argomento operatore argomento ...*

Mostra sull'uscita standard il risultato dell'espressione. Gli argomenti devono essere numeri interi (senza virgola), gli operatori sono:

(i) operatori aritmetici: `+`, `-`, `*`, `/`, `%` (resto);

(ii) operatori di relazione: `=`, `!=` (diverso), `<`, `>`, `<=`, `>=`.

Per gli operatori di relazione, il comando stampa "`1`" (e restituisce 0) se l'espressione è vera e "`0`" (e restituisce 1) se è falsa. Attenzione a proteggere `*`, `<` etc. con apici (singoli o doppi), in modo che non vengano interpretati dalla shell.
Esempi:

```
$ expr 10 / 2 + 5 - 4
6
$ expr 5 '*' 100
500
$ expr 5 * 100
expr: syntax error
```

Nell'ultimo caso, il comando ha segnalato un errore di sintassi perchè il carattere `*` è stato trasformato, dalla shell, nella lista dei file contenuti nella directory corrente, prima di essere passato al comando stesso.

```
$ expr 2 '<' 3
1
$ if expr 2 '<' 3 >/dev/null; then echo "ok"; else echo "?!"; fi
ok
```

grep

Mostra le linee che contengono una stringa.

`grep` [`-v`] *stringa file1 file2 ...*

Legge tutte le linee di *file1*, *file2*, etc. (oppure dell'ingresso standard, se nessun file viene specificato) e mostra sull'uscita standard solo quelle che contengono *stringa*. Se viene specificato più di un file, le linee mostrate sarano precedute dal nome del file da cui sono estratte, seguito da "`:`".

Con l'opzione `-v`, mostra le linee che *non* contengono *stringa*.
Esempi:

```
$ grep aa fileB
dddcaa  aaaccab
$ grep aa fileB fileC
fileB:dddcaa  aaaccab
fileC:aaa
```

```
fileC:aaa
$ grep -v aa fileC
bbb
bbb
bbb
ccc
$ ls | grep A
fileA
```

head

Mostra la prima parte di un file.

`head` [`-n`*numero*] [*file*]

Mostra sull'uscita standard le prima *numero* (10 se `-n`*numero* non viene specificato) linee di *file* (o dell'ingresso standard, se non viene specificato un file).
Esempi:

```
$ head -n3 fileC
aaa
bbb
bbb
$ ls | head -n2
fileA
fileB
```

kill

Manda un segnale ad uno o più processi.

`kill` *pid1 pid2* ...

Invia un segnale ai processi con identificatore *pid1*, *pid2*, etc. Normalmente, ciò causa la terminazione dei processi coinvolti.
Esempi: Creiamo il sorgente C di un programma che esegue un loop infinito:

```
$ echo "int main() { while(1) ; }" > loop.c
```

Lo compiliamo, ottenendo l'eseguibile `a.out` nella directory corrente:

```
$ cc loop.c
$ ls -l a.out
-rwxr-xr-x  1 giuseppe users 7236 May 24 10:48 a.out
```

Lo mandiamo in esecuzione in background, quindi ne otteniamo il pid e, infine, lo uccidiamo:

```
$ ./a.out &
$ ps | grep a.out
 9955 pts/1    00:00:04 a.out
$ kill  9955
9955 Terminated              ./a.out
$ ps | grep a.out
```

ln

Crea collegamenti tra file.

`ln` [`-s`] *file collegamento*

Crea un collegamento, di nome *collegamento*, al file *file*. Con l'opzione `-s`, il collegamento è di tipo simbolico.
Esempi:

```
$ ls -il fileA fileB
11 -rw-r--r--  1 giuseppe users  8 May 23 12:02 fileA
12 -rw-r--r--  1 giuseppe users 26 May 23 12:02 fileB
$ ln fileA hard
$ ls -il fileA hard
11 -rw-r--r--  2 giuseppe users 8 May 23 12:02 fileA
11 -rw-r--r--  2 giuseppe users 8 May 23 12:02 hard
$ rm fileA
$ ls -il fileA hard
ls: fileA: No such file or directory
11 -rw-r--r--  1 giuseppe users 8 May 23 12:02 hard
$ cat hard
aa
bbbb
$ ln -s fileB soft
$ ls -il fileB soft
12 -rw-r--r--  1 giuseppe users 26 May 23 12:02 fileB
15 lrwxrwxrwx  1 giuseppe users  5 May 23 19:25 soft -> fileB
$ rm fileB
$ ls -il fileB soft
ls: fileB: No such file or directory
15 lrwxrwxrwx  1 giuseppe users 5 May 23 19:25 soft -> fileB
$ cat soft
cat: soft: No such file or directory
```

Si noti che, quando un collegamento punta a un file non esistente, si ottiengono, in genere, messaggi di errore poco chiari: `cat` dice che `soft` non esiste, ma, in realtà, è il file puntato da `soft` a non esistere.

ls

Elenca il contenuto di una directory.

`ls` [`-ail`] [*directory*]

`ls` [`-il`] *file1 file2* ...

Nella prima forma, mostra il contenuto della directory *directory*. Con l'opzione `-a`, mostra anche i file e le directory il cui nome comincia per "`.`". Con l'opzione `-i`, mostra anche il numero di i-node di ogni elemento contenuto. Con l'opzione `-l` mostra, per ogni elemento contenuto, i permessi, il numero di collegamenti,

il proprietario, il gruppo, la dimensione, la data di ultima modifica e il nome dell'elemento (seguito da "`->`" e il percorso dell'elemento puntato, nel caso di collegamenti simbolici). Se il parametro *directory* viene omesso, si intende la directory corrente.

Nella seconda forma, mostra le informazioni richieste (`-i` e/o `-l`) solo per i file specificati, se esistono. Se non viene specificata nessuna opzione, si limita a mostrare, uno per linea, i nomi dei file, così come scritti, se esistono (e non necessariamente nello stesso ordine in cui sono scritti).
Esempi:

```
$ ls -l
total 4
-rw-r--r--  1 giuseppe users  8 May 23 12:02 fileA
-rw-r--r--  1 giuseppe users 26 May 23 12:02 fileB
-rw-r--r--  1 giuseppe users 24 May 23 12:02 fileC
-rw-r--r--  1 giuseppe users 45 May 23 12:02 fileD
$ ls -i
11 fileA
12 fileB
13 fileC
14 fileD
$ ls -a
.
..
fileA
fileB
fileC
fileD
$ ls -il
total 4
11 -rw-r--r--  1 giuseppe users  8 May 23 12:02 fileA
12 -rw-r--r--  1 giuseppe users 26 May 23 12:02 fileB
13 -rw-r--r--  1 giuseppe users 24 May 23 12:02 fileC
14 -rw-r--r--  1 giuseppe users 45 May 23 12:02 fileD
$ ls -ail
total 7
     2 drwxr-xr-x   2 giuseppe users 1024 May 23 12:13 .
173329 drwxr-xr-x  36 giuseppe users 2000 May 23 19:25 ..
    11 -rw-r--r--   1 giuseppe users    8 May 23 12:02 fileA
    12 -rw-r--r--   1 giuseppe users   26 May 23 12:02 fileB
    13 -rw-r--r--   1 giuseppe users   24 May 23 12:02 fileC
    14 -rw-r--r--   1 giuseppe users   45 May 23 12:02 fileD
$ mkdir dir1
$ cp file* dir1
$ ls dir1
fileA
```

```
fileB
fileC
fileD
$ ls -ail dir1
total 6
15 drwxr-xr-x  2 giuseppe users 1024 May 23 19:25 .
 2 drwxr-xr-x  3 giuseppe users 1024 May 23 19:25 ..
16 -rw-r--r--  1 giuseppe users    8 May 23 19:25 fileA
17 -rw-r--r--  1 giuseppe users   26 May 23 19:25 fileB
18 -rw-r--r--  1 giuseppe users   24 May 23 19:25 fileC
19 -rw-r--r--  1 giuseppe users   45 May 23 19:25 fileD
$ ls fileA fileC non_esistente dir1/fileD
ls: non_esistente: No such file or directory
dir1/fileD
fileA
fileC
```

mail

Invia mail.

`mail` [`-s`*oggetto*] *destinatario1 destinatario2 ...*

Invia una mail a tutti i destinatari specificati, con oggetto pari a *oggetto* (se specificato). Il contenuto della mail viene letto dall'ingresso standard.
Esempi:

```
$ echo "Ciao" | mail -s "Un saluto" giuseppe
```

mesg

Accetta/non accetta messaggi sul terminale.

`mesg y`

`mesg n`

Esempi:

```
$ tty
/dev/pts/1
$ ls -l /dev/pts/1
crw-------  1 giuseppe tty 136, 1 May 24 10:46 /dev/pts/1
$ mesg y
$ ls -l /dev/pts/1
crw--w----  1 giuseppe tty 136, 1 May 24 10:46 /dev/pts/1
```

mkdir

Crea directory.

`mkdir` *directory1 directory2 ...*

Crea le directory passate per parametro. Segnala un errore se una directory esiste già, oppure se non esiste una parte del percorso che dovrebbe portare alla directory da creare.
Esempi:

```
$ mkdir dir1/dir2
mkdir: cannot create directory ‘dir1/dir2’: No such file or directory
$ mkdir dir1
$ mkdir dir1/dir2
$ ls -i | grep dir1
15 dir1
$ ls -ia dir1
15 .
 2 ..
16 dir2
$ ls -ia dir1/dir2
16 .
15 ..
$ mkdir dir1
mkdir: cannot create directory ‘dir1’: File exists
```

mv

Sposta o rinomina file.

`mv` *file1 file2*

`mv` *file1 file2 ... directory*

Nella prima forma, rinomina *file1* in *file2*. Se esiste già un file di nome *file2*, questo viene prima cancellato.

Nella seconda forma, sposta tutti i file *file1*, *file2*, etc., nella directory *directory*, che deve esistere già.
Esempi:

```
$ ls -il fileA
11 -rw-r--r--  1 giuseppe users 8 May 23 12:02 fileA
$ mv fileA nuovo
$ ls -il fileA nuovo
ls: fileA: No such file or directory
11 -rw-r--r--  1 giuseppe users 8 May 23 12:02 nuovo
$ mkdir dir1
$ mv file* dir1
$ ls -il
total 2
15 drwxr-xr-x  2 giuseppe users 1024 May 23 19:25 dir1
11 -rw-r--r--  1 giuseppe users    8 May 23 12:02 nuovo
$ ls -il dir1
total 3
```

```
12 -rw-r--r--  1 giuseppe users 26 May 23 12:02 fileB
13 -rw-r--r--  1 giuseppe users 24 May 23 12:02 fileC
14 -rw-r--r--  1 giuseppe users 45 May 23 12:02 fileD
```

nl

Numera le linee dei file.

`nl` *file1 file2* ...

Mostra sull'uscita standard il contenuto dei file *file1*, *file2* etc., aggiungendo una colonna che contine un numero (progressivo) di linea. Se, e solo se, nessun file viene specificato, esegue la sua operazione sull'ingresso standard.
Esempi:

```
$ nl fileA fileB
     1  aa
     2  bbbb
     3  ccbbb acc
     4  dddcaa  aaaccab
$ ls -a | nl
     1  .
     2  ..
     3  fileA
     4  fileB
     5  fileC
     6  fileD
```

ps

Mostra (una parte dei) processi correnti

`ps` [`aux`]

Chiamato senza argomenti, mostra tutti i processi che appartengono all'utente che lo ha invocato e che sono associati al terminale corrente (in prima approssimazione, tutti i processi ancora in esecuzione, creati nella sessione corrente).

Con le tre opzioni `aux` mostra tutti i processi in esecuzione (anche quelli degli altri utenti), in formato esteso.
Esempi:

```
$ ps
  PID TTY          TIME CMD
11473 pts/5    00:00:00 bash
11893 pts/5    00:00:00 examples.sh
11902 pts/5    00:00:00 examples.sh
11904 pts/5    00:00:00 expand
11905 pts/5    00:00:00 sed
11910 pts/5    00:00:00 ps
```

Per ogni processo viene mostrato l'identificatore (PID), il terminale di controllo (TTY), il tempo di CPU usato fin'ora (TIME) e il programma che il processo stà eseguendo (CMD).

```
$ ps aux | head
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1448   292 ?        S    May19   0:00 init [3]
root         2  0.0  0.0      0     0 ?        SN   May19   0:00 [ksoftirqd/0]
root         3  0.0  0.0      0     0 ?        S<   May19   0:01 [events/0]
root         4  0.0  0.0      0     0 ?        S<   May19   0:00 [khelper]
root         9  0.0  0.0      0     0 ?        S<   May19   0:00 [kthread]
root        20  0.0  0.0      0     0 ?        S<   May19   0:00 [kacpid]
root        18  0.0  0.0      0     0 ?        S<   May19   0:02 [vesafb]
root       107  0.0  0.0      0     0 ?        S<   May19   0:06 [kblockd/0]
root       166  0.0  0.0      0     0 ?        S    May19   0:16 [kswapd0]
```

In questo caso vengono mostrate, per ogni processo, molte più informazioni. Senza scendere nei dettagli, riconosciamo il proprietario (USER), PID, TIME (stesso significato di prima), COMMAND (come CMD) e START che ci dice quando il processo ha iniziato la sua esecuzione. In particolare, riconosciamo il processo di PID 1 (init), che è il primo ad essere creato (all'avvio del sistema) e che resta sempre in esecuzione fino a quando il sistema non viene spento.

pwd

Stampa la directory corrente.

`pwd`

Esempi:

```
$ pwd
/home/giuseppe/Corsi/SE/se/unix/prove
$ mkdir dir1
$ cd dir1
$ pwd
/home/giuseppe/Corsi/SE/se/unix/prove/dir1
$ cd ..
$ pwd
/home/giuseppe/Corsi/SE/se/unix/prove
```

rm

Cancella file.

`rm` *file1 file2 ...*

Elimina i file passati come argomento. Segnala un errore se uno (o più) dei file da cancellare non esiste.
Esempi:

```
$ ls -il fileA
11 -rw-r--r--  1 giuseppe users 8 May 23 12:02 fileA
$ rm fileA
$ ls -il fileA
ls: fileA: No such file or directory
```

```
$ rm fileA
rm: cannot remove ‘fileA’: No such file or directory
```

rmdir

Rimuove directory.

`rmdir` *directory1 directory2 ...*

Elimina le directory specificate. Le directory devono essere vuote (devono contenere solo i collegamenti "`.`" e "`..`"). Segnala un errore se una directory da eliminare non esiste.
Esempi:

```
$ mkdir dir1
$ ls
dir1
fileA
fileB
fileC
fileD
$ rmdir dir1
$ ls
fileA
fileB
fileC
fileD
$ mkdir dir1
$ cp fileA dir1
$ rmdir dir1
rmdir: ‘dir1’: Directory not empty
$ rm dir1/fileA
$ rmdir dir1
```

sed

Stream EDitor.

`sed` *comando file1 file2 ...*

Applica *comando* a tutte le righe di tutti i file, una alla volta, (o all'ingresso standard, se non viene specificato nessun file) e mostra il risultato sull'uscita standard. I comandi applicabili sono svariati. Ci limitiamo a ricordare il comando di sostituzione, che ha la sintassi: `s` *delim sorgente delim destinazione delim opzioni*, dove *delim* è un carattere qualsiasi, che non deve apparire in *sorgente* e *destinazione*, e serve a delimitare i campi, mentre *opzioni* può essere nullo, oppure la lettera "`g`". Il comando ordina la sostituzione della stringa *sorgente* con la stringa *destinazione*, in ogni riga. Se non viene specificata l'opzione `g`, solo la prima occorrenza di *sorgente* (in ogni riga) verrà sostituita. Con l'opzione `g`, verranno sostituite tutte le occorrenze. Comando deve essere un'unico argomento, quindi, se contiene spazi, va protetto con apici singoli o doppi.
Esempi:

```
$ sed s/cc/Z/ fileB
Zbbb acc
dddcaa  aaaZab
$ sed s/cc/Z/g fileB
Zbbb aZ
dddcaa  aaaZab
$ ls | sed s/file//
A
B
C
D
```

seq

Mostra una sequenza di numeri.

`seq` [*primo* [*incremento*]] *ultimo*

Mostra sull'uscita standard una sequenza di numeri, uno per linea, che va da *primo* (1 se assente) a *ultimo*, con incrementi di *incremento* (1 se assente).
Esempi:

```
$ seq 3
1
2
3
$ seq 10 13
10
11
12
13
$ seq 13 -1 10
13
12
11
10
```

sort

Ordina uno o più file.

`sort` [`-rn`] *file1 file2* ...

Mostra in uscita le linee di tutti i file specificati (o le linee lette dall'ingresso standard, se nessun file viene specificato) ordinate secondo l'ordinamento ASCII crescente (quindi, all'incirca alfabeticamente). Con l'opzione `-r`, le linee vengono ordinate al contrario. Con l'opzione `-n`, il comando si aspetta di trovare un numero all'inizio di ogni riga, e ordina le righe seguendo l'ordinamento numerico.
Esempi:

```
$ sort fileA fileC
aa
aaa
aaa
bbb
bbb
bbb
bbbb
ccc
$ sort -r fileB
dddcaa  aaaccab
ccbbb acc
$ nl fileC | sort -rn
     6  ccc
     5  aaa
     4  bbb
     3  bbb
     2  bbb
     1  aaa
```

tail

Mostra l'ultima parte di un file.

`tail` [`-n`*numero*] [*file*]

Mostra sull'uscita standard le ultime *numero* (10 se `-n`*numero* non viene specificato) linee di *file* (o dell'ingresso standard, se non viene specificato un file). *Esempi*:

```
$ tail -n3 fileC
bbb
aaa
ccc
$ ls | tail -n2
fileC
fileD
```

test

Controlla i tipi di file e confronta espressioni.

`test` *stringa operatore stringa*

`test` *operatore file*

Il comando esegue dei test su delle stringhe (prima forma) o dei file (seconda forma) e quindi restituisce 0 (se il test è vero) o 1 (se il test è falso).

Nella prima forma, *operatore* può essere `=` oppure `!=` (diverso).

Nella seconda forma, *operatore* può essere:

`-f` *file* esiste ed è un file normale;

`-d` *file* esiste ed è una directory;

`-c` *file* esiste ed è un dispositivo a caratteri;

`-b` *file* esiste ed è un dispositivo a blocchi;

`-L` *file* esiste ed è un collegamento simbolico;

`-e` *file* esiste (senza dire di che tipo è);

`-s` *file* esiste ed ha dimensione maggiore di 0.

Esempi:

```
$ if test pippo = pluto; then echo uguali; fi
$ if test -f fileA; then echo file regolare; fi
file regolare
$ mkdir dir1
$ if test -d dir1; then echo directory; fi
directory
```

touch

Modifica la data di ultima modifica di uno o più file.

`touch` *file1 file2* ...

La data (e ora) di ultima modifica di ogni file diventerà uguale alla data (e ora) corrente. Se uno (o più) dei file non esiste, verrà creato (vuoto, cioè con dimensione pari a 0).
Esempi:

```
$ ls -l fileA
-rw-r--r--  1 giuseppe users 8 May 23 12:02 fileA
$ touch fileA
$ ls -l fileA
-rw-r--r--  1 giuseppe users 8 May 23 19:25 fileA
$ touch nuovo
$ ls -l nuovo
-rw-r--r--  1 giuseppe users 0 May 23 19:25 nuovo
```

tr

Trasforma o cancella caratteri.

`tr` *caratteri1 caratteri2*

`tr -s` *caratteri*

Nella prima forma, i parametri *caratteri1* e *caratteri2* devono essere due sequenze di caratteri (per semplicità, supponiamo che le due sequenze contengano lo stesso numero di caratteri). Il comando legge il suo ingresso standard, trasforma ogni carattere che compare in *caratteri1* nel corrispondente carattere in *caratteri2* e scrive il risultato sull'uscita standard. I caratteri che non compaiono in *caratteri1* vengono lasciati inalterati.

Nella seconda forma, elimina le sequenze di caratteri ripetuti: ogni volta che uno dei caratteri elencati nel parametro *caratteri* si presenta ripetuto più volte di seguito, il comando `tr` ne mostra un'unica copia.

Le sequenze di caratteri possono essere abbreviate con la notazione *c1*`-`*c2*, che rappresenta tutti i caratteri da *c1* a *c2*. Ad es., scrivere `a-d` è come scrivere `abcd` e scrivere `xyza-cA-C` è come scrivere `xyzabcABC`.
Esempi:

```
$ tr b z < fileA
aa
zzzz
$ tr a-z A-Z < fileB
CCBBB ACC
DDDCAA  AAACCAB
$ tr -s c < fileB
cbbb ac
dddcaa  aaacab
$ ls -l | tr -s ' '
total 4
-rw-r--r-- 1 giuseppe users 8 May 23 12:02 fileA
-rw-r--r-- 1 giuseppe users 26 May 23 12:02 fileB
-rw-r--r-- 1 giuseppe users 24 May 23 12:02 fileC
-rw-r--r-- 1 giuseppe users 45 May 23 12:02 fileD
```

tty

Mostra il nome del terminale associato all'ingresso standard.

`tty`

Esempi:

```
$ tty
/dev/pts/1
```

uniq

Rimuove le linee duplicate.

`uniq` [`-c`] [*ingresso*]

Legge il file *ingresso* (o l'ingresso standard, se nessun file viene specificato), e, ogni volta che incontra una sequenza di linee identiche, ne invia una sola all'uscita standard (le linee che già in *ingresso* non facevano parte di una sequenza di linee identiche, passano inalterate).

Con l'opzione `-c`, mostra, accanto ad ogni linea, il numero di linee uguali a quella che erano presenti, in sequenza, nel file di ingresso.
Esempi:

```
$ uniq fileC
aaa
bbb
aaa
ccc
$ sort fileC | uniq
aaa
bbb
ccc
$ sort fileC | uniq -c
      2 aaa
      3 bbb
      1 ccc
```

wc

Stampa il numero di linee, parole e caratteri in uno o più file.

`wc` [`-lwc`] *file1 file2* ...

Normalmente, vengono mostrate sull'uscita standard tutte e tre le informazioni (nell'ordine) per ogni file richiesto, seguita da una linea con i totali su tutti i file. Se viene specificata una (o più) delle opzioni `-l` (linee), `-w` (parole), `-c` (caratteri), verranno mostrate solo le informazioni corrispondenti. Se, e solo se, non viene specificato alcun nome di file, il comando legge il suo ingresso standard.
Esempi:

```
$ wc fileA fileB
 2  2  8 fileA
 2  4 26 fileB
 4  6 34 total
$ wc -l fileA fileB
 2 fileA
 2 fileB
 4 total
$ wc -c fileA
8 fileA
$ ls -l fileA
-rw-r--r--  1 giuseppe users 8 May 23 12:02 fileA
$ ls | wc -l
4
```

who

Mostra gli utenti collegati.

`who`

Mostra gli utenti collegati, il nome del loro terminale e la data e l'ora in cui si sono collegati.
Esempi:

```
$ who
giuseppe :0           May 23 11:47
giuseppe pts/0        May 23 11:47
giuseppe pts/1        May 23 11:47
giuseppe pts/2        May 23 11:47
giuseppe pts/3        May 23 11:47
```

Notare che uno stesso utente può apparire collegato più volte, soprattutto in presenza di un terminale grafico (indicato con `:0` nell'esempio precedente), in quanto ogni finestra di shell è associata ad uno pseudo-terminale.

write

Manda un messaggio ad un altro utente.

`write` *utente*

Invia un messaggio all'utente *utente*. L'utente deve essere colleggato e deve accettare i messaggi. Il messaggio viene letto dall'ingresso standard.