Breve dispensa sui comandi della shell Unix

Versione del 6 aprile 2022

Si ricorda che nella maggior parte dei sistemi Unix sono disponibile le pagine di manuale per molti dei comandi. Per consultare la pagina di manuale di un comando, per esempio `ls`, è sufficiente invocare il comando `man`:

```
$ man ls
```

In particolare il comando `man` stesso ha un manuale, invocabile con `man man`.

Navigazione tra directory

`pwd` – Stampa la directory corrente

Il comando `pwd` stampa la directory corrente:

```
$ pwd
/home/rickastley/
```

`cd` – Cambia la directory

Il comando `cd` permette di cambiare la directory in cui ci si trova. Esso può essere usato in diversi modi:

- Con un path relativo indica come spostarsi rispetto alla directory corrente.
- Con un path assoluto indica il path in cui spostarsi
- Senza argomenti riporta alla directory home dell'utente

`ls` – Visualizza il contenuto della directory

Il comando `ls` visualizza i file e le directory contenute nella directory corrente. Di default i file il cui nome inizia con un punto non sono visibili (questo include `.` e `..`, rispettivamente la directory corrente e la directory padre).

Alcune opzioni utili sono:

- `-a`. Visualizza tutti i file inclusi quelli il cui nome inizia con un punti.
- `-l`. Visualizza informazioni aggiuntive (e.g., dimensione, permessi, etc.) per ogni file.
- `-R`. Esegue il comando `ls` ricorsivamente su tutte le sotto-directory.

Creazione, rimozione e spostamento di file

`touch` – Creazione di file

Il comando `touch` seguito dal nome di un file ne aggiorna la data di accesso e modifica. Nel caso il file non esista esso viene creato. Il file così creato è vuoto (0 byte di dimensione).

`cp` – Copia di file e directory

Il comando `cp` viene utilizzato per copiare file e directory. Per esempio

```
$ cp file1 file2
```

copia `file1` in un nuovo file chiamato `file2`.

Alcune opzioni utili:

- `-R` effettua la copia ricorsivamente (a volte indicata come una *deep copy*)

`mv` – Muovere e rinominare file e directory

Il comando `mv` viene usato per muovere (ed eventualmente rinominare) file e directory. Per esempio:

```
$ mv file1 directory2/file2
```

sposta il file `file1` all'interno della directory `directory2` rinominandolo `file2`. Come si può notare spostare e rinominare un file sono operazioni simili svolte entrambe dallo stesso comando.

`rm` – Rimozione di file e directory

Il comando `rm` serve a rimuovere i file e le directory passate come argomento. Una opzione spesso usata è `-r`, che indica che la rimozione deve essere effettuata ricorsivamente, quindi eliminando tutto il contenuto di una directory prima di eliminare la directory stessa.

`mkdir` – Creazione di directory

Il comando `mkdir` viene utilizzato per creare una nuova directory (vuota) col nome fornito come argomento. Per esempio

```
$ mkdir dir3
```

crea una directory vuota chiamata `dir3` nella directory corrente.

`rmdir` – Cancellazione di directory

Il comando `rmdir` rimuove la directory *vuota* passata come argomento.

Lettura del contenuto dei file

`cat` – Stampare il contenuto di un file

Il comando `cat` prende come argomento il nome di un file e stampa a terminale (su *standard output*) il suo contenuto.

`head` e `tail` – Vedere le prime / ultime righe di un file

I comandi head e tail scrivono su standard output le prime 10 righe (`head`) o le ultime 10 righe (`tail`) di un file passato come argomento. Il numero di righe è modificabile con l'opzione `-n [count]`, che permette di specificare il numero di righe da mandare in output.

`less` – Vedere il contenuto di un file in modo paginato

Il comando `less` (e il suo predecessore `more`) permette di visualizzare il contenuto di un file in maniera paginata, ovvero visualizzando una singola schermata del contenuto del file e permettendo di spostarsi avanti e indietro nella visualizzazione. Si può uscire con `q`.

Manipolazione di dati

`sort` – Ordinare le righe di un file

Il comando `sort` permette di ordinare le righe contenute in un file. Di default l'ordinamento è lessicografico in ordine crescente (e.g., `a < aa < ab < b`, etc.). È possibile modificare il tipo di ordinamento con le seguenti opzioni:

- `-n`. Effettua un ordinamento numerico invece che lessicografico.
- `-r`. Inverte la direzione dell'ordinamento (ordine decrescente invece che crescente).

`wc` – Contare caratteri, parole e righe

Il comando `wc` serve a contare il numero di linee, parole e bytes contenuti in un file. Di default tutti questi valori vengono inviati in output. È possibile scegliere solo alcuni di essi con opzioni specifiche:

- `-l`. Conta le linee.
- `-w`. Conta le parole.
- `-c`. Conta i byte.

`cut` – Estrarre campi da un file

Il comando `cut` serve a lavorare su dati in formato tabulare. Si consideri, per esempio, un file in cui ogni riga è della forma:

```
Nome, Cognome, Età, Via, Città, CAP
```

Se volessimo raccogliere solamente alcune di queste informazioni (e.g., l'età e il CAP) sarebbe necessario lavorare a livello di colonne. Per questo è possibile usare `cut`, passando come opzioni quali colonne recuperare e che carattere indica la separazione tra le diverse colonne (di default il carattere "tab"):

- `-f [colonne]`. Indica quali colonne preservare. Le colonne sono numerate a partire da 1; colonne multiple sono selezionabili inserendo più numeri separati da virgola.
- `-d [delimitatore]`. Indica quale carattere è il separatore tra le colonne.

Nel nostro esempio per ottenere solo età e CAP il comando sarebbe:

```
$ cut -f 3,6 -d ','
```

`grep` – Individuare righe che rispettano un pattern

Il comando `grep` permette di individuare quali righe di un file contengono un certo pattern (e.g., un parola). Il primo argomento passato a grep è il pattern (può essere una singola parola o una espressione regolare), il secondo argomento è il file in cui effettuare la ricerca. Una opzione utile è `-v`, che stampa solo le righe che *non* contengono il pattern (i.e., inverte il risultato della ricerca).

Redirezione e pipe

Ogni comando nella shell inizia con tre file aperti dai quali è possibile leggere e scrive:

- *standard input*. Da dove viene letto l'input (e.g., con `getchar()`).
- *standard output*. Dove viene inviato l'output (e.g., con `printf()`).
- *standard error*. Dove vengono inviati i messaggi di errore.

Lo standard input viene solitamente "preso" da tastiera, mentre lo standard output e lo standard error stampano su terminale. È però possibile redirigere lo standard output su file usando `>` dopo un comando. Per esempio

```
ls -l > contenuto.txt
```

reindirizza tutto quando viene mandato in standard output da `ls -l` nel file `contenuto.txt` (sovrascrivendo il contenuto precedente del file). La redirezione può anche *aggiungere* contenuto ad un file se effettuata con `>>` invece di `>`.

È anche possibile collegare lo standard output di un comando allo standard input di un altro comando tramite una pipe `|`. In questo modo è possibile comporre comandi in sequenza. Per esempio

```
head -n 20 file.txt | grep Ciao
```

passa come input a `grep Ciao` l'output di `head -n 10 file.txt`, quindi il risultato finale di questo comando sarà di produrre in output le righe tra le prime 20 del file `file.txt` che contengono la sequenza di caratteri `Ciao`.

Editor di testo VI

L'editor di testo `vi` ed i suoi successori, come `vim` o `neovim`, sono editor di testo modali le cui origini risalgono alla versione originale di `vi` nel 1976.

Differentemente dagli editor di testo che normalmente si incontrano, vi e i suoi discendenti sono modali. Questi significa che in ogni dato istante l'editor si trova in uno tra due stati:

- *modalità di inserimento*. In questa modalità i caratteri corrispondenti ai tasti premuti sono inseriti all'interno del file. Questo è il normale comportamento di molti editor.
- *modalità di comando*. In questa modalità i tasti premuti non provocano l'inserimento di caratteri nel file, ma sono interpretati come comandi. Vi all'avvio è in questa modalità.

Vi e, in particolare, vim e neovim, sono editor molto complessi e potenti, vedremo però qui solo alcuni comandi di base:

- `:w`. Salva il contenuto del file.
- `:q`. Esce da vi. Può essere forzato con `:q!` nel caso il file non sia stato in precedenza salvato. Si può salvare e uscire combinando i comandi in `:wq`.
- `i`. Entra in modalità inserimento.
- `dd`. Cancella la riga in cui si trova il cursore.
- `dw`. Cancella la parola su cui si trova il cursore.
- `x`. Cancella il carattere su cui si trova il cursore.
- Il tasto `ESC` permette di passare dalla modalità inserimento alla modalità di comando.

I permessi in Unix

Ogni file e directory in Unix ha associati un utente e un gruppo proprietari e una serie di permessi riguardanti le operazioni che si possono svolgere sul file o directory:

- **Read (Lettura)**. Il permesso di lettura consente di leggere il contenuto di un file o, nel caso delle directory, di vedere quali file sono contenuti all'interno.
- **Write (Scrittura)**. Il permesso di scrittura consente di modificare il contenuto di un file. Nel caso delle directory è permesso creare, rinominare o cancellare i file contenuti all'interno (se il permesso di execute è anch'esso concesso).
- **Execute (Esecuzione)**. Il permesso di esecuzione consente di eseguire un file (e.g., uno script della shell). Per le directory permette di accedere al contenuto della directory ma non di vedere i nomi dei file contenuti (i.e., è necessario sapere i nomi dei file per accedervi se il permesso di lettura non è anch'esso concesso).

Generalmente i permessi sono presentati da comandi come `ls -l` come una sequenza di 9 caratteri da leggersi a gruppi di tre:

Utente	Gruppo	Altri
`rwx`	`rwx`	`rwx`

dove la presenza della lettera indica che il permesso è concesso e la presenza di `-` indica che non lo è. Per esempio `rwxr-xr--` indica che l'utente proprietario può leggere, scrivere e eseguire il file, gli appartenenti al gruppo proprietario possono leggere ed eseguire il file (ma non scrivere), mentre tutti gli altri utenti possono solo leggerlo.

I permessi possono essere anche espressi numericamente usando un numero di tre cifre in base ottale (base 8). A ognuno dei permessi è assegnato un valore:

- Read: 4
- Write: 2
- Execute 1

E la cifra ottale corrispondente si ottiene sommando i valori dei permessi che sono concessi. Per esempio `r-x` avrà valore $4 + 1 = 5$. Usando una cifra per i permessi dell'utente, una per i permessi del gruppo e, infine, una per i permessi di tutti gli altri, è possibile esprimere i permessi di un file con tre cifre. Per esempio `rwxr-xr--` verrebbe espresso come `754`.

`chmod` – cambiare i permessi dei file Il comando `chmod` permette di cambiare i permessi di un file (solo l'utente proprietario del file o il superutente possono effettuare questa modifica). La sintassi è `chmod [permessi] [nome_file]` dove i permessi possono essere espressi con tre cifre ottali.