Shell Scripting

Daniel Falbo --- danielfalbo@engineer.com

Introduzione

A nessuno piace svolgere **compiti noiosi e ripetitivi**. Mentre alcuni di questi compiti ancora necessitano di essere ripetuti manualmente, altri, sopratutto se ci sono **macchine** coinvolte, sono **automatizzabili**. Al giorno d'oggi, tra remote working e smart learning, molti di noi potrebbero **salvare tempo prezioso quotidianamente** automatizzando compiti che ripetutamente svolgono con i propri **computers**.

> 1. **Esercizio per il lettore**: *elenca almeno 3 procedure che svolgi quotidianamente con il tuo telefono o computer, pensa (anche solo intuitivamente, ignorando per ora l'implementazione pratica) a come potrebbero essere automatizzate e stima quanto tempo risparmieresti ogni giorno.*

Se parliamo di programmatori, uno degli strumenti con il quale passano più del loro tempo è il terminale **[1]**. L'automatizzazione di comandi ripetuti al prompt di un terminale, è detta **shell scripting**, e consiste nel creare (ed eventualmente eseguire) file contenenti le istruzioni che si vogliono automatizzare e l'interprete che dovrebbe eseguire queste operazioni.

> 2. **Esercizio per il lettore programmatore**: *svolgi l'* esercizio 1 *ma limitati ad elencare solo procedure ripetitive che svolgi nel tuo terminale o nel tuo IDE.*

L'implementazione di questi **scripts** è leggermente differente in base al sistema operativo del computer sul quale si sta operando. Quando si tratta di shell scripting, possiamo dividere i sistemi operativi in 2 grandi categorie:

- i sistemi operativi **UNIX-like**, ispirati al sistema operativo UNIX, come ad esempio macOS,
- e i sistemi operativi della famiglia **DOS**, ispirati, per l'appunto, al **DOS**, come ad esempio Windows.

---

[1] Bisogna fare distinzione tra i terminali e le shell: i terminali sono gli emulatori che permettono all'utente di visualizzare graficamente i messaggi e gli errori della shell (STDOUT ed STDERR nei sistemi unix-like) e rispondere inviando altro testo (STDIN nei sistemi unix-like), mentre le shell sono interfacce fra l'utente e i servizi del sistema operativo. Questo articolo approfondisce riguardo STDIN, STDOUT ed STDERR con riferimenti al codice sorgente del kernel di linux.

DOS e Windows

Partendo da MS-DOS all'inizio degli anni 80 fino ad arrivare al recente Windows 10, i sistemi operativi della famiglia DOS sono i più diffusi tra i computer comuni. Al giorno d'oggi in Windows il framework di scripting più comodo da utilizzare è PowerShell, decisamente più moderno e versatile del caro, vecchio MS-DOS.

Sistemi UNIX-like

Storicamente

Nel 1979 la settima versione di unix fu rilasciata con, come shell predefinita, la Bourne shell, spesso chiamata semplicemente "sh". Da quel giorno sh rimase una shell di riferimento per i sistemi unix-like e ancora oggi è inclusa in molti sistemi.

> 3. **Esercizio**: se possiedi un computer con un sistema operativo unix-like, come macOS o tutte le distribuzioni linux, verifica se `sh` è presente nel tuo sistema eseguendo `which sh` nel terminale (se vuoi, puoi leggere il manuale di istruzioni del comando `which` lanciando `man which`).

La Bourne shell è una shell molto semplice, basilare. Tra i comandi disponibili **[2]** abbiamo `cd`, che prende come argomento il percorso di una directory e si sposta in quella directory, `pwd`, che stampa a video il percorso completo della directory corrente, `echo`, che stampa a video tutti gli argomenti che riceve, ma anche blocchi condizionali come i `while` loops, i `for` loops o gli `if ; then` statements.

Nei sistemi unix-like, in genere gli script iniziano con una linea che contiene un asterisco (hashtag), un punto esclamativo (bang) e il percorso all'interprete dello script, chiamata comunemente shebang line. Per convenzione, su tutti i sistemi unix-like il percorso della bourne shell è `/bin/sh`, quindi la prima riga di un nostro shell script sarebbe

```
#! /bin/sh
```

mentre per altri interpreti dei quali non siamo sicuri del percorso, si utilizza

```
#! /usr/bin/env <interprete>
```

assumendo che `env` sia in `/usr/bin/` come da convenzione.

4. **Esercizio**: come mai? Utilizza `man env` per capire cos'è `env` e perchè utilizzare `#! /usr/bin/env <interprete>` come shebang line ha senso.

Anche se nel caso della sh l'# corrisponde al carattere che inizia i commenti, la shebang line è valida e non comporta problemi neanche per i linguaggi dove i commenti non iniziano per # finchè è esattamente la prima linea del file.

Tra le migliaia di linee di codice del kernel di linux, open-source e disponible all'indirizzo github.com/torvalds/linux, possiamo osservare che la prima linea di codice eseguita quando viene caricato uno script

```
if ((bprm->buf[0] != '#') || (bprm->buf[1] != '!'))
    return -ENOEXEC;
```

controlla i primi due byte o caratteri del file e si assicura che il primo non sia diverso da `'#'` ed il secondo non sia diverso da `'!'`.

Scripting moderno

In realtà con l'esercizio 3 ho mentito: anche se `which sh` mostra con successo il percorso che porta ad `sh` sulla tua macchina, probabilmente non hai la vera `sh`. È comune infatti al giorno d'oggi rimpiazzare `sh` con un collegamento ad una shell più moderna ma retrocompatibile, come la Bourne again shell ("shell rinata"), spesso chiamata semplicemente "bash".

Scriviamo ed eseguiamo il nostro primo script insieme

Per scrivere uno script, la prima cosa di cui abbiamo bisogno è un editor di testo, per questo articolo utilizzerò `vi`, un editor incluso nella maggior parte dei sistemi unix-like, che probabilmente hai anche tu (controlla lanciando `which vi`, in caso `vi` non sia installato, qualunque altro editor di testo che già conosci va benissimo).

Lo script che stiamo per scrivere prenderà due argomenti, una località di partenza ed una destinazione, ed aprirà il nostro web browser sui risultati di Google Voli relativi.

Dal terminale, il comando `vi voli` aprirà `vi` con un nuovo file, chiamato `voli`, il nostro script. Una volta aperto l'editor inseriamo la shebang line, digitando prima `i` per passare dalla `normal mode` di `vi` alla `insert mode`, per poter scrivere, e poi digitando la nostra shebang line, `#! /bin/sh`. Il comando utilizzato per aprire un link nel web browser predefinito è `open` su macOS e `xdg-open` sulla maggior parte delle distribuzioni di Linux. Utilizzeremo `$1` ed `$2` per prendere il primo ed il secondo argomento e li includeremo nella richiesta a google. Quindi digitiamo invio in `vi` per andare alla riga successiva ed aggiungiamo

```
#! /bin/sh

open "https://www.google.com/search?q=Voli%20da%20$1%20a%20$2"
```

se siamo su macOS, oppure

```
#! /bin/sh

xdg-open "https://www.google.com/search?q=Voli%20da%20$1%20a%20$2"
```

se siamo su una distribuzione Linux. Dopodichè premiamo <esc> per uscire dalla insert mode di `vi`, salviamo il file digitando `:write` (e premendo invio) ed usciamo da `vi` digitando `:quit`. Adesso lanciando `ls` vediamo che tra i file contenuti nella nostra directory corrente c'è anche il nostro `voli`. Lo rendiamo eseguibile lanciando

```
chmod +x voli
```

ed infine lo eseguiamo, lanciando, per esempio,

```
./voli Bari Parigi
```

Complimenti! Hai appena scritto ed eseguito il tuo primo shell script.

---

[2] Per i più curiosi, il comando `help` stampa a video tutti i comandi built-in ed il comando `man` prende come argomento un altro comando e stampa a video il manuale delle istruzioni di quel comando (ad esempio `man echo` stampa a video le istruzioni del comando `echo`).