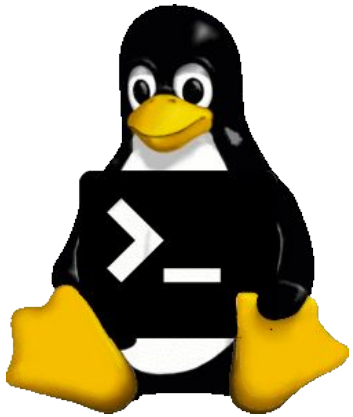
Linux in a Shell

Giuseppe Piccolo

CLUB SVILUPPATORI



Chi sono

- BSc Computer Science @ Uniba
- Software Developer @ ApuliaSoft
- Membro del Club degli Sviluppatori - Puglia
- *nix User :)

Outline

- La shell
- La shell in *nix
- Bash
- Caratteristiche di Bash
- Scripting con bash

La shell

“La **shell** (detta anche interprete dei comandi), in informatica, è la parte di un sistema operativo che permette agli utenti di interagire con il sistema stesso, impartendo comandi e richiedendo l'avvio di altri programmi. Insieme al kernel costituisce una delle componenti principali di un sistema operativo. Il suo nome (dall'inglese shell, guscio) deriva dal fatto che questa componente viene considerata l'involucro, **la parte visibile** del sistema ed è dunque definibile come **l'interfaccia** utente o il programma software che la rende possibile.”

Wikipedia

GNU/Linux

- Linux è uno dei kernel adottati dal sistema operativo GNU.
- GNU/Linux è una distribuzione del sistema operativo GNU composta dal kernel linux e dagli strumenti messi a supporto dello stesso, sviluppati dalla Free Software Foundation (FSF)
- Il nome è tuttora oggetto di controversia
- Per comodità utilizzeremo **Linux** come alias di **GNU/Linux**
- Ad oggi sul mercato esistono centinaia di disrtibuzioni linux (ubuntu, debian, redhat, ec…)

La shell nel mondo *nix - sh

Nel 1971 Ken Thompson rilascia la prima versione della Thompson shell (sh). Rispetto alle shell moderne questa shell era piuttosto rudimentale: aveva il supporto alle pipe, alcune strutture di controllo di base (come if/then e goto) ed le *wildcard* sui nomi dei file.

Ispirata alla shell del sistema operativo Multics basata sul modello RUNCOM (run command - comando eseguibile).

Nei sistemi unix-like (come linux) molti file di configurazione terminano con suffisso *rc* in memoria dell'antenato delle shell unix RUNCOM .

La shell nel mondo *nix - bourne shell

La Bourne shell è una riscrittura completa della shell originale sh ad opera di Stephen Bourne. Distribuita per la prima volta nel 1979 con la versione 7 di UNIX, di fatto sostituiva la shell originale di Thompson.

Questa versione portava numerose caratteristiche come:

- here document
- sostituzione dei comandi
- variabili d’ambiente
- strutture di controllo avanzate (es. loop)

La shell nel mondo *nix - La bourne shell (2)

Tradizionalmente la bourne shell si avvia attraverso il programma *sh*, sito nella cartella */bin*. Tuttavia oggi giorno sh non è nient'altro che un alias per una delle sue numerose alternative.

La shell nel mondo *nix - La bourne shell (2)

- Almquist shell (ash)
- Bourne-Again shell (bash)
- Korn shell (ksh)
- Debian Almquist shell (dash)
- Public domain Korn shell (pdksh)
- MirBSD Korn shell (mksh)
- Z shell (zsh)
- Busybox

Bash

Bourne-**A**gain **sh**ell è una shell scritta da Brian Fox nel contesto del progetto GNU come alternativa free ed open source alla bourne shell. La sua prima versione risale al 1989 e, al giorno d'oggi, è la shell di default di molte distribuzioni Linux e di Mac OSX.

E' disponibile anche per le piattaforme Microsoft Windows attraverso il tool *cgywin*. A partire dalla versione Anniversary Update di Windows 10 la shell bash è stata integrata in modo "nativo" nel sistema operativo.

Bash (2)

Spesso viene chiamata anche bourne shell dal nome dell’autore (Stephen Bourne) della shell da cui è stata ispirata bash.

L'espansione dell'acronimo bash, Bourne again shell, letteralmente significa un'altra shell Bourne, ma Bourne again ha la stessa pronuncia di born again, cioè rinata creando la definizione finale shell rinata.

Bash (3)

Tecnicamente bash è un interprete di comandi che permette all’utente di comunicare con il sistema operativo attraverso una serie di funzioni predefinite (built in) o di **eseguire programmi e script**

Caratteristiche di bash

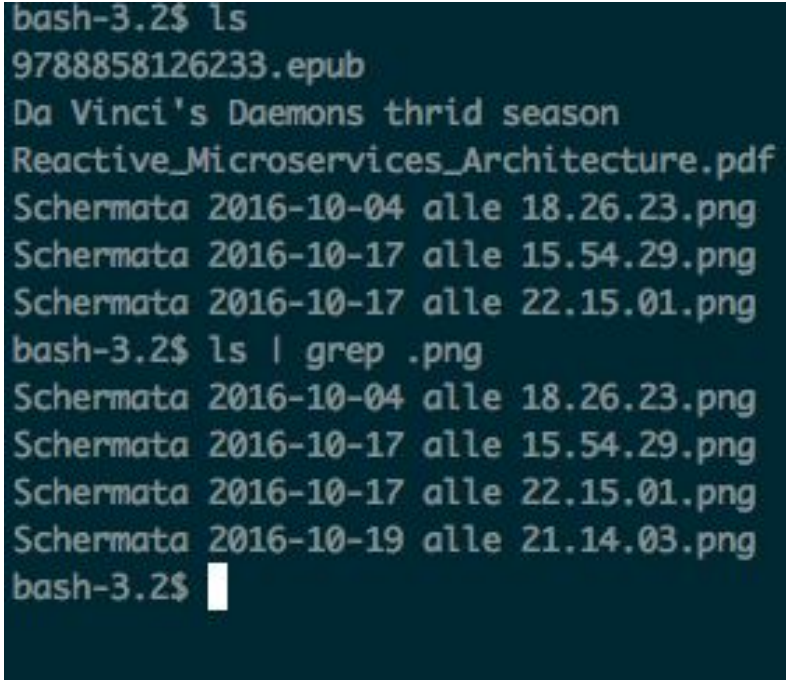
Redirezione

Bash è in grado di eseguire la redirezione dell’input e dell’output dei programmi eseguiti su di essa in modo da eseguire più programmi in sequenza creando una vera e propria pipeline software dove l’output del programma precedente diventa l’input del programma successivo

Pipeline

Filtrare i file in una cartelle per estensione

```
ls | grep .png
```



Pipeline (2)

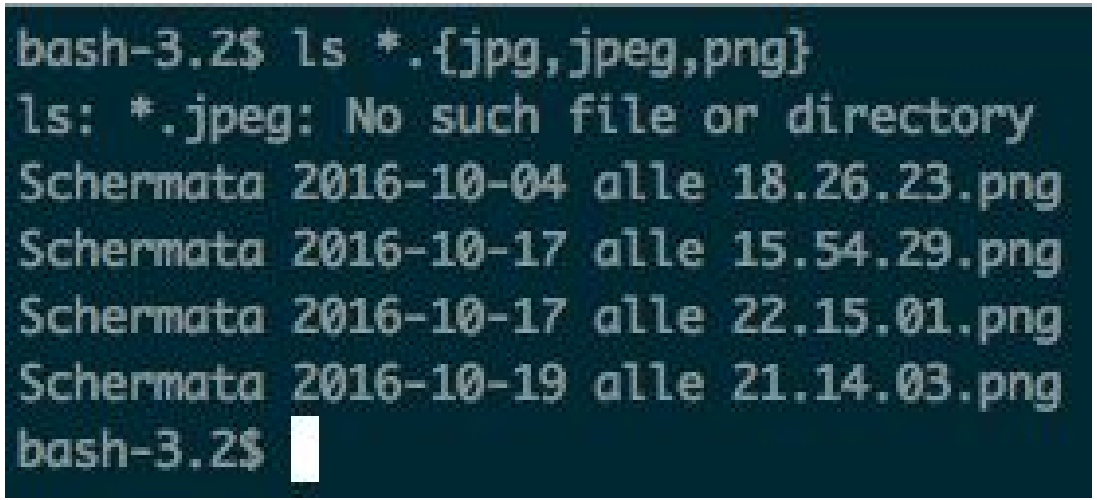
Redirezione su file

```
curl www.google.it > google.html
```

Brace expansion

La brace expansion è una caratteristica che permette di generare tutte le possibili combinazioni tra una stringa ed un insieme di alternative.

```
$ echo a{p,c,d,b}e
ape ace ade abe
$ echo {a,b,c}{d,e,f}
ad ae af bd be bf cd ce cf
```



Brace expansion (2)

La brace expansion può essere utilizzata anche per esprimere sequenze

```
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
$ echo file{1..4}.txt
file1.txt file2.txt file3.txt file4.txt
$ echo {a..e}
a b c d e
$ echo {1..10..3}
1 4 7 10
$ echo {a..j..3}
a d g j
```

```
bash-3.2$ touch file{1..4}.txt
bash-3.2$ ls
file1.txt       file2.txt       file3.txt       file4.txt
bash-3.2$
```

Gestione dei processi

La shell bash ha due modi con il quale esegue i programmi: **batch mode** (sequenziale) e **concurrent mode** (concorrente)

Se digitiamo i comandi separati da un **;**

```
command1; command2
```

stiamo eseguendo i comandi in modalità **batch**, ovvero l'esecuzione di command2 avverrà dopo l'esecuzione di command1 sia che esso fallisca o vada a buon fine

Gestione dei processi (2)

Se digitiamo

```
command1 &command2
```

stiamo eseguendo i comandi in modalità **concorrente.** command1 viene eseguito in **background** ed il controllo della shell passa a command2 che viene eseguito in **foreground**. E’ possibile sospendere command2 con ctrl+z.

Gestione dei processi (3)

Un programma sospeso può essere rievocato utilizzando il comando **fg** (se al momento della sospensione si trovata in foreground) o **bg** (se si trovava in background). L'id del processo in background è contenuto nella variabile **$!**

La shell può essere sincronizzata per attendere la fine di un job in background utilizzando il comando **wait**, seguito dal *process id* (**pid**) del processo da attendere. se non viene passato il pid, wait forza la shell ad attendere che tutti i job in background terminino.

Esecuzione condizionale

In bash è possibile eseguire comandi sequenza in modo che l'esecuzione del primo condizioni l'esecuzione del secondo. Ad esempio se digitiamo

```
command1 && command2
```

l'esecuzione di command2 avverrà se e solo se l'esecuzione di command1 va a buon fine. Analogamente

```
command1 || command2
```

indica che l'esecuzione di command2 avviene se command1 fallisce.

Configurazione di bash

Alla partenza, bash tenta di caricare automaticamente una serie di file:

- se si tratta di una shell di login, bash tenta di caricare:
  1. il file "/etc/profile"
  2. solo il primo tra i file ".bash_profile", ".bash_login" e ".profile" collocati nella home directory dell'utente e che risulti essere disponibile;
  3. il file ".bashrc" collocato nella home directory dell'utente;

All'uscita di una shell interattiva di login, bash carica il file .bash_logout collocato nella home dell'utente (se disponibile).

- se si tratta di una shell interattiva non di login, bash tenta di caricare il solo file ".bashrc" collocato nella home directory dell'utente.

Configurazione di bash (2)

L'utente può modificare i file nella propria home directory per personalizzare il proprio ambiente di lavoro.

Quando bash è avviata non interattivamente (ad esempio per eseguire uno script di shell) essa controlla se esiste la variabile d'ambiente **BASH_ENV** o **ENV** ed in caso positivo carica il file specificato dal valore della variabile (se bash è avviata in modalità POSIX controlla solo ENV). Procede quindi ad eseguire il resto (script o altro).

Scripting in bash

Scripting in Bash

Bash mette a disposizione un linguaggio di scripting piuttosto semplice, che permette di svolgere compiti piuttosto complessi, astraendo la realizzazione all'interno di **script** che possono diventare dei **programmi eseguibili** da bash, avvalendosi del supporto di caratteristiche proprie dei linguaggi di programmazione più evoluti come le **strutture di controllo** e le **variabili**.

Anatomia di uno script bash

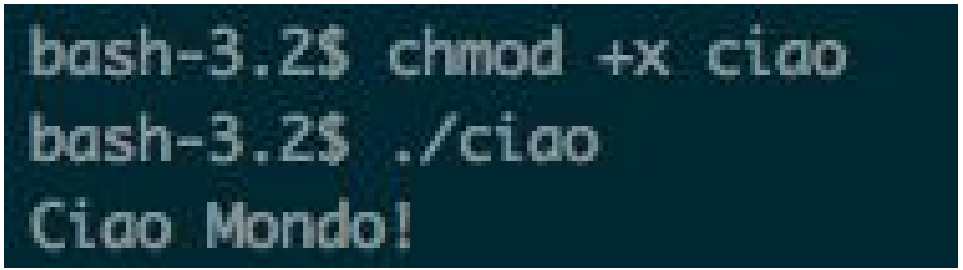
```
#!/bin/bash
tar -vzcf /tmp/home.tgz ~
```

Il seguente script permette di creare un archivio in "/tmp/my-backup.tar.gz" contenente l'intera propria cartella personale (tenendo conto che "~" rappresenta "/home/proprionomeutente"):

Anatomia di uno script (2)

per rendere eseguibile lo script creato in precedenza è sufficiente assegnare al file contenente lo script i permessi di esecuzione con il comando **chmod**

```
chmod +x nomefile
```



La $PATH in bash

tutti gli script eseguibili in bash possono essere eseguiti direttamente dalla riga di comando (esattamente come le utility di base) a patto che essi siano eseguibili e siano ubicati in uno dei percorsi indicati nella variabile d'ambiente $PATH

```
bash-3.2$ echo $PATH
/Users/pino/anaconda/bin:/usr/local/sbin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:
/sbin:/opt/X11/bin:/usr/local/share/dotnet
bash-3.2$
```

Le strutture condizionali

Il linguaggio di scripting di bash supporta le espressioni condizionali, ovvero espressioni del tipo:

```
se <si verifica questa condizione> allora
     <esegui queste istruzioni>
altrimenti
     <esegui queste altre istruzioni>
```

Le strutture condizionali (2)

Nel linguaggio bash la sintassi per esprimere le strutture condizionali è la seguente:

```
if expression; then
     …
else
     …
fi;
```

Le strutture condizionali - esempi

Confronto tra due file

```
#!/bin/bash
if cmp file_a file_b  &>/dev/null; then
   echo "I File a e b sono identici."
else
   echo "I File a e b sono diversi."
fi
```

Le strutture condizionali - esempi (2)

Controlla l’esistenza di un file

```
#!/bin/bash
if [ -f $1 ]; then
    echo "Il file $1 esiste."
else
    echo "Il file $1 non esiste o non è un file"
fi
```

Le strutture di controllo - esempi (3)

```
#!/bin/bash
echo "Scrivi qualcosa e premi Invio:"
read linea
case "$linea" in
   uno)
      echo "Hai digitato 'uno'"
      ;;
   *)
      echo "Hai digitato qualcosa che non conosco"
      ;;
esac
```

I cicli

Bash supporta le strutture iterative, ovvero particolari strutture che consentono di eseguire istruzioni comuni su collezioni di dati. Bash supporta tre tipi di cicli:

- for
- while
- until

I cicli - for

Il ciclo di tipo **for** differisce dal tipico ciclo for che ritroviamo negli altri linguaggi di programmazione. In bash il form permette di iterare soltanto sulle stringhe (collezione di caratteri)

for - esempio

```
#!/bin/bash
    for i in $( ls ); do
        echo item: $i
    done
```

In questo esempio, stampiamo i contenuti della cartella in cui ci troviamo.

La variabile **i** conterrà i vari valori rappresentanti i nomi dei file contenuti nella cartella.

La parola chiave **done** indica la terminazione del blocco di istruzioni del ciclo.

I cicli - while

Il ciclo while si comporta esattamente come negli altri linguaggi di programmazione, ovvero esegue iterativamente un blocco di istruzioni fintantoché la condizione specificata è rispettata

while - esempio

```
#!/bin/bash
        COUNTER=0
        while [  $COUNTER -lt 10 ]; do
            echo The counter is $COUNTER
            let COUNTER=COUNTER+1
        done
```

In questo esempio, viene stampato il valore di COUNTER fintanto che esso risulta essere minore o uguale di 10.

I cicli - until

Il ciclo until si comporta esattamente come negli altri linguaggi di programmazione, ovvero esegue iterativamente un blocco di istruzioni fintantoché la condizione specificata è rispettata. A livello semantico il suo funzionamento è identico al ciclo while.

until - esempio

```
#!/bin/bash
        COUNTER=20
        until [  $COUNTER -lt 10 ]; do
            echo COUNTER $COUNTER
            let COUNTER-=1
        done
```

In questo esempio, viene stampato il valore di COUNTER fintanto che esso risulta essere minore o uguale di 10, con la differenza che in questo caso COUNTER viene inizializzato a 20, pertanto verranno stampati i numeri al contrario

Altri comandi

Una trattazione esaustiva delle funzionalità e dei comandi di bash è disponibile al seguente indirizzo

https://www.gnu.org/software/bash/manual/html_node/

un ulteriore aiuto è dato dall'help in linea, richiamabile attraverso il comando **man**.

Grazie per l’attenzione