Documentazione Introduzione Scripting

Radu Andries

20 Marzo 2013

Part I
Linguaggi di scripting

1 I Linguaggi di scripting

Ci sono vari linguaggi di scripting disponibili al sysadmin, ognuno con le proprie particolarità. Ve ne descriverò alcuni

1.1 Shell script

I shell script sono di solito script per bash (le altre shell sono usate meno) e sono il linguaggio di scripting più usato nell'ambiente GNU/Linux. È facile da imparare ed ha a disposizione tutti i tool CLI (inclusi programmi installati) del sistema operativo. Spesso anche programmi grafici possono essere comandati da terminale, rendendo i shell script molto utili.

1.2 Perl

PERL (Practical Extraction and Reporting Language) è un linguaggio molto potente. Si dice che per qualsiasi task amministrativo ci sia un oneliner in perl, cioè un comando di una linea. Ha una sintassi molto flessibile e ha un'enormità di moduli per assistervi nei vostri task amministrativi.

1.3 Python

Python è poco usato per amministrare il sistema ma resta un linguaggio molto potente. La sua particolarità è nel usare tabulazioni e spazi per determinare l'appartenenza ad un blocco di codice. Anche python ha tanti moduli disponibili.

2 Gli script

Gli script sono file testuali che si possono eseguire. La prima riga dei script è una riga speciale, indica al sistema che interprete usare per eseguire lo script.

Si chiama shebang ed è del formato *#!interprete.* Esempi:

```
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/python
```

Part II
Bash

3 Bash Basics

La shell di solito viene usata in modo interattivo. In questo caso si possono usare delle combinazioni di tasti per controllare l'esecuzione dei programmi. Ecco alcune combinazioni utili:

- CTRL-Z Ferma il programma temporaneamente. Può essere ripreso con il comando "fg" o mandato in background con "bg"
- CTRL-C Invia segnale di arresto al programma
- CTRL-D Invia il carattere EOF

Bash come tutti i programmi ha a disposizione 3 "file" standard.
STDIN - lettura dell'input
STDOUT - scrittura a schermo
STDERR - scrittura a schermo di errori

3.1 Operazioni su file

- cd - entra nella cartella data come argomento
- mv - rinomina/muove un file
- rm - cancella un file
  - "-r" ricorsivo
  - "-f" forza
- rmdir - cancella una directory
- ls - visualizza i file della cartella corrente oppure quella passata come argomento
  - "-la" fa vedere tutte le directory e file
  - "-h" visualizza le dimensioni del file in modo "umano"
- touch - crea un file se non esiste.

3.2 Hello World

echo stampa a schermo i suoi argomenti.

```
#!/bin/bash
echo Hello World
```

3.3 Argomenti agli script

Gli argomenti in bash vengono passati nelle variabili $1,$2 e così via. Tutti gli argomenti sono messi in $@. I doppi apici permettono di includere variabili di bash.

```
#!/bin/bash
echo "Hello ${1}"
```

3.4 Lettura e redirezioni

Per leggere file si usa cat. Ha un unico parametro, il nome del file da leggere. Si può redirigere l'output o l'input di un programma:

- ">" redirige l'output di un programma in un file
- ">>" redirige l'output di un programma in un file aggiungendo alla fine del file invece di sovrascrivere
- "<" legge lo standard input da un file
- "|" redirige l'output del programma nello standard input di un altro programma

Il seguente programma lancia cat per leggere "ilmiofile" e inviarlo a "./mioscript2". Il ./ indica che lo script è nella cartella corrente. L'output di mioscript viene scritta sul file ilmiofile

```
#!/bin/bash
cat ilmiofile | ./mioscript > ilmiooutput
```

Si può redirigere lo STDERR a STDOUT insieme a file oppure in file diversi:

- &>miofile redirige STDERR e STDOUT a miofile
- 2>miofile redirige solo STDERR a file

Esempio:

```
#!/bin/bash
programmaconlog 2>errori.log >output.log &
programmaconlog &>outputunificato.log
```

3.5 Background

Per eseguire in background un programma basta aggiungere "&" dopo il comando. Per farlo tornare in foreground c'è il programma "fg".

4 Bash Constructs

4.1 Lettura STDIN

Per leggere da stdin si usa il comando "read" accetta come parametro il nome della variabile dove scrive la stringa letta.

4.2 Condizioni if

In bash if è molto "cattivo". Richiede che siano rispettati gli spazi. il formato è:

```
if [[ espressione ]]; then
   codice
else
   codice
fi
```

Per quanto riguarda l'espressione è implicito l'uso del comando "test". Ecco alcuni esempi:

```
#Esegue codice se esiste miofile
if [[ -e miofile ]]; then
  codice
fi
```

4.3 Loop while

Esegue un blocco di codice finché l'espressione diventa false. È molto simile di sintassi a if:

```
while [[ espressione ]]; do
   codice
done
```

4.4 Loop for

Esegue un blocco di codice un numero finito di volte. Ci sono due modi. Il primo cicla variabile (il separatore è $IFS)

```
for name in variabile ;  do
   codice
done
```

Il secondo invece funziona come l'omonimo di C:

```
for (( expr1 ; expr2 ; expr3 )) ;
do
    codice
done
```

5 Programmi importanti

5.1 Pager

Il pager è un comando che permette di leggere output lunghi (ad esempio cat di un file enorme) in modo più comodo. Ci sono 2 che sono più usati, "more" e "less". Permettono di fare scroll con le frecce e di leggere direttamente file passati come argomento. Less ha più funzionalità di more (paradosso!) ed è considerato leggermente migliore.

5.2 man

Permette di ricercare documentazione relativa ad un programma. Ad esempio "man bash" visualizza il manuale di bash nella console.

5.3 sleep

Alcune volte può essere utile aspettare un po' di tempo prima di continuare con l'esecuzione, ad esempio è utile in un ciclo while infinito per non eseguire lo stesso codice mille volte.

```
#!/bin/bash
echo Ciao. Aspetto 5 secondi
sleep 5
echo Addio mondo crudele
```

5.4 cut

Permette di estrarre colonne da STDIN. Con il parametro -d si imposta il delimitatore. Con -f si seleziona la colonna.

```
#Visualizza i moduli del kernel caricati al momento (1a colonna nel file /proc
cut -f1 -d" " /proc/modules
```

5.5 uniq & sort

Servono rispettivamente a rimuovere duplicati e ordinare una lista. operano su STDIN e file. Le righe devono essere già ordinate purche uniq funzioni

5.6 tee

tee è un comando utilissimo per redirigere in un log l'output di un programma e nello stesso tempo seguirlo sullo schermo. Si passa come parametro il file dove salvare l'output.

5.7 wc

Conta le righe, le parole e la lunghezza totale del testo passato in STDIN

5.8 head & tail

Servono per visualizzare righe all'inizio o alla fine di un file, stream da STDIN. Accettano il parametro -n per dire quante righe visualizzare.

```
head -n3 /tmp/openvpn.log
tail -n20 /tmp/openvpn.log
```

tail ha una variante chiamata tailf (ma aggiungendo il parametro -f anche tail si comporta allo stesso modo) che permette di seguire l'output su un file.

Part III
Altri Programmi

5.9 top,htop

Sono dei comandi per vedere in modo interattivo lo stato del sistema

5.10 watch

Watch esegue un comando in un intervallo prefissato con l'opzione n. Ad esempio "watch -n 1 – ls /dev" esegue ogni secondo "ls /dev" e visualizza il risultato sullo schermo.

5.11 file

Permette di individuare la tipologia di file con cui abbiamo a che fare. Legge l'inizio del file per capirne il contenuto, non importa l'estensione.

5.12 grep

È un programma molto complesso che permette di applicare delle Regular Expression ad ogni riga di testo e visualizzare il contenuto in funzione dei parametri e se la riga corrisponde al RegExp

Dato che la più semplice RegExp è una parola da ricercare può essere facilmente usato per verificare la presenza di una certa sequenza di caratteri in un file.

```
#Conta le righe che contengono la parola pippo in file
cat file | grep pippo | wc -l
```

5.13 awk & sed

Sono dei comandi molto potenti, anche se difficili da usare. awk è un linguaggio di scripting per il processing di testo. sed invece permette di applicare RegExp su STDIN. Essendo così complessi si consiglia la lettura di man per impararli a usare.

5.14 screen

GNU Screen è un programma complesso che permette di avere più interfacce di terminale nella stessa finestra. Ha davvero troppe feature per essere negli obiettivi di questo corso, ma una base è necessaria. Spesso screen viene usato per mantenere dei programmi in esecuzione anche quando un utente non è collegato al server. Si hanno più finestre che si possono controllare via shortcut. Inoltre si possono creare più sessioni (istanze) diverse di screen allo stesso tempo. La sessione finisce quando viene chiuso screen oppure quando tutti i terminali all'interno sono stati chiusi

Ecco una piccola cheat sheet dei principali comandi, con esempi.

```
#lancia screen creando la sessione "adm"
screen -S adm
#si riconnette alla sessione "adm"
screen -r adm
#disconnette tutti gli altri client e si ricollega a "adm"
screen -d -r adm

#Alcuni comandi all'interno di screen
C-a c - Crea una nuova finestra
C-a 1 - Va alla finestra 1
C-a d - si stacca da screen (screen continua ad andare in background)

#Split finestre
C-a | - Separa in due la finestra verticalmente
C-a S - Separa in due orizzontalmente
C-a Tab - passa alla prossima finestra
```

Part IV
Editor di testo

Da tempi immemori c'è la battaglia degli editor di testo sutto GNU/Linux e prima ancora Unix. I più usati sono:

6 Nano

nano è un editor di testo molto semplice da usare ma con davvero poche feature. Avendo di default le istruzioni sempre visibili è perfetto per chi ha iniziato ad usare il terminale da poco.

7 Vim

Vim è un editor particolare che ha speciali modalità per l'inserimento, per la selezione e per i comandi. È un editor molto estendibile, tanto è che molti lo usano come se fosse un IDE.

8 Emacs

Emacs fu creato da Richard Stallman, la persona che ha ideato il sistema operativo GNU e gran parte dei programmi base, incluso il compilatore gcc. È un editor molto potente ed estendibile.