Corso facilitato di bash scripting

By CasertaGLUG per informazioni contattare l'autore casertaglug-owner@autistici.org

Prima Parte:

Parte 1. Introduzione

La shell è un interprete di comandi. Molto più che una semplice interfaccia tra il kernel del sistema operativo e l'utilizzatore, è anche un vero e proprio potente linguaggio di programmazione. Un programma di shell, chiamato *script*, è uno strumento semplice da usare per creare applicazioni "incollando" insieme chiamate di sistema, strumenti, utilità e file binari (eseguibili). Uno script di shell può utilizzare virtualmente l'intero repertorio di comandi, utility e strumenti UNIX. Se ciò non fosse abbastanza, i comandi interni della shell, come i costrutti di verifica ed i cicli, forniscono ulteriore potenza e flessibilità agli script. Questi si prestano eccezionalmente bene a compiti di amministrazione di sistema e a lavori ripetitivi e di routine, senza l'enfasi di un complesso, e fortemente strutturato, linguaggio di programmazione.

Capitolo 1. Programmare la shell.

L'autore ci descrive che imparare praticamente l'uso della programmazione della shell (prompt dei comandi di Unix) è utile diventare esperti amministratori di rete ma non solo. A questo punto quando noi avviamo la nostra distribuzione preferita, accade che ci sono dei cicli di avvio, tutta una serie di servizi che vengono caricati all'avvio del sistema operativo in memoria.

Questi servizi in particolare la shell vengono caricati da questo file (/etc/rc.d) compreso anke il ripristino del sistema e l'attivazione dei servizi. Imparare a scrivere degli script non è difficile e la sintassi è semplice e chiara. Premetto che per chi non è ancora pratico dello script si consiglia di vedere il corso di Linux di base.

Comunque programmare la shell non comporta molte regole di gestione rispetto agli altri programmi di compilazione come il (C, C++, Java).

E' utile per eseguire una serie di comandi che noi spesso usiamo di solito, ma che se li integriamo in un file eseguibile ci torna utile per non imputarli manualmente come il classico e odiatissimo file batch dell'MSDOS in Windows.

Nb: Per creare un file di script è necessario imputare nel prompt bash (es. [linux.room]#: nano –w nomescript.sh) da qui potete iniziare a scrivere il vostro script. Dopo tutto ciò è necessario

impostare i permessi di esecuzione ammettendo che stiate su un nome utente root, il comando seguente da digitare (chmod u+rx nomescript) dopo di che vedete cosa succede.

Se volete vedere gli esempi dovete andare sulla guida completa del manuale originale.

Quando non usare gli script i shell:

- Non va utilizzato quando richiede un uso troppo intenso di risorse del computer.
- Per operazioni matematiche complesse in virgola mobile ecc. Si consiglia di usare (C,C++, java per cose più complesse).
- Non usare in applicazioni complesse.
- Ecc ecc.

Questo linguaggio è portabile e facilmente gestibile non è adatto alla programmazione di applicazioni complesse come i C e il C++ e altri e oltretutto rapido.

Dizionario

Bash = Bourne-Again shell (Variante del Sistema Unix)

Shell = prompt unix

Sharp = #

Che estensione ha un eseguibile di shell? Essenzialmente 2 e cioè (mioscript.sh, mioscript.bash).

In poche parole, uno script non è nient'altro che un elenco di comandi di sistema posti in un file. Oltretutto si risparmia la fatica per ridigitare una particolare sequenza di comandi tutte le volte che necessario. Ad esempio se io voglio fare la lista della connessione utente e successivamente cancellare i file log, dovrò prima digitare il comando con un opzione chiamata (flag) e specificare l'ip, andare nella directory del file log e cancellare i dati registrati. Tutto questo lo posso inglobare in file eseguibile di sistema shell, eseguirlo e il gioco è fatto.

Piccolo esempio pratico

```
  1 # Cleanup
  2 # Da eseguire come root, naturalmente.
  3
  4 cd /var/log
  5 cat /dev/null > messages
  6 cat /dev/null > wtmp
  7 echo "Log cancellati."
```

Questo è imputato con #! Punto esclamativo

```
  1 #!/bin/sh
  2 #!/bin/bash
  3 #!/usr/bin/perl
  4 #!/usr/bin/tcl
  5 #!/bin/sed -f
  6 #!/usr/awk -f
```

Adesso osservate solo la parte strutturale ma non considerate i comandi inseriti in sequenza.

Ora il simbolo specificato (#) serve ad inserire un testo che in fase di esecuzione non viene considerato dall'elaboratore può essere sfruttato come una descrizione pratica per ricordare quali passaggi si devono effettuare oppure la descrizione di un comando. In alcune distro si inserisce il simbolo (#!) con punto esclamativo che vale 2 byte.

Non vado oltre con gli esempi per non confondervi con le troppe istruzioni specificate in questo capitolo.

Nella vostra distribuzione vedrete che le directory anno una, sua struttura ed una sua linearità. Questo è chiamato standard **Posix**.(Posizione Standard in gergo delle directory dello Unix)

Per eseguire uno script bisogna digitare (bash mioscript) oppure in alternativa (sh mioscript) ma questa modalità è può essere rischiosa perché disabilita la lettura di una dipendenza shell chiamata (stdin) all'interno dello script. E' molto più conveniente rendere lo script eseguibile direttamente con il comando (chmod).

Si può fare con:

chmod 555 mioscript (da a tutti gli utenti il permesso di lettura/esecuzione)

Oppure con:

chmod +rx mioscript (come il precedente)

chmod u+rx mioscript (che attribuisce solo al proprietario dello script il permesso di leggere e scrivere).

Dopo avergli attribuito i permessi possiamo digitare **(./mioscript**) nella directory che state lavorando, ad esempio supponiamo che vi troviate in (/home/nick/scripts) nella shell va eseguito inserendo un punto e un backslash iniziale e il nome del file premete invio e il programma comincia fare tutto da solo.

Se ci sono errori di sintassi il terminale vi avvisa con un messaggio di errore generico tipo (command nont fournd).

Ora se volete che questo piccolo script che abbiamo creato lo vogliate inserire nella cartella di sistema lo spostate nella directory con il comando (cp /home/nick/scripts/mioscript.sh /usr/local/bin). Copia il file nella cartella di sistema così da poterlo successivamente eseguirlo come un comando unico direttamente da un terminale in qualsiasi directory voi vi troviate.

Quando si inizia uno script è bene specificare la directory della shell e cioè (/bin/sh oppure /bin/bash).

Un altro esempio sull'impostazione corretta dello script

```
1 #!/bin/bash ←specificare qui
  2
  3 echo "Parte 1 dello script." <- Abilita e disabilita il prompt di linea di
comando parte iniziale dello script
  4 a=1 "Variabile"
  5
  6 #!/bin/bash
  7 # Questo *non* eseguirà un nuovo script.
  8
  9 echo "Parte 2 dello script."
 10 echo $a  # Il valore di $a è rimasto 1.
```

Adesso se io voglio visualizzare un file (Leggimi) dalla shell io normalmente digito il comando (Leggimi.txt | more) mi visualizzerà il file leggimi per pagine separate seguite da un invio dell'utente passando pagina per pagina. Nello script dovrà essere specificato con (#!/bin/more).

Posix significa Portable Operating System Interface

Le esercitazioni dal manuale originale sono le seguenti.

Alcuni espediente per le variabili

Noi possiamo dichiarare delle variabili con un nome a nostro piacimento per evitare di pacificare un valore come una directory molto lunga.

Le variabili di dichiariamo così (nomevar = valore [directory, numero intero ecc])
Tipo

```
1 VAR_1 = /home/randomlife/scripts/mioscripts.sh
```

Per richiamare una variabile evitando così di riscrivere il percorso completo è così ($VAR_1). Questo pio potrebbe essere sfruttato in una condizione in programmazione oppure in quale altra specifica.

Ricordatevi che quando create una file o richiamante un file con lettere maiuscole e minuscole è (case sensitive) nel senso che lo Unix fa distinzione tra maiuscole e minuscole ad esempio se io voglio andare in una dir che si chiama (/Home) è diverso da (/home) scrivetelo così come lo vedete.

Ovviamente posso anche specificare 5 funzioni su una stessa riga di comando specificando un punto e virgola ogni fine comando esempio.

```
root:/# mount /dev/hdc1 /mnt/hd2; cp /home/randomlife/script/mioscriptsh
/mnt/hd2; umount /dev/hdc /mnt/hd2.
```

Esercizi preliminari

1. Gli amministratori di sistema spesso creano degli script per eseguire automaticamente compiti di routine. Si forniscano diversi esempi in cui tali script potrebbero essere utili.
2. Scrivere uno script che all'esecuzione visualizzi l'ora e la data, elenchi tutti gli utenti connessi e fornisca il tempo di esecuzione uptime del sistema. Lo script, quindi, dovrà salvare queste informazioni in un file di log.