Guida basilare di Scripting Bash - matrobriva

Short description of this text:Una guida che insegna le basi dello scripting bash.

Upload date:2012-03-19 **Original date:**0000-00-00

Matrobriva

Guida di scripting shell

VERSIONE BASE

Introduzione

Siete affascinati dal mondo della programmazione? Non avete tempo o non riuscite ad imparare linguaggi complessi come il C? Utilizzate GNU/Linux o altri sistemi UNIX? Allora probabilmente questo fa proprio al caso vostro. In queste pagine imparerete a sfruttare al massimo le potenzialità del linguaggio scripting integrato nella shell di Unix: è facile da imparare e ci si può fare di tutto. Basta chiacchiere. Partiamo.

Cos'è uno script

Uno script è semplicemente un file di testo ASCII contenente dei comandi, precisamente quelli che vengono lanciati in un normale terminale. Per questo le possibilità di uno script sono praticamente illimitate. Chiaramente, in scripting userete comandi che raramente userete in shell, quali echo o read.

Hello world

Da sempre è tradizione usare sempre come primo esempio di un linguagio di programmazione lo scriptino “Hello world”. Vediamo come farlo in bash-scripting. Apriamo un editor di testi semplici (Kwrite, Gedit, nano, vim, emacs...) e digitiamo la seguente riga:

```
echo "hello world"
```

Poi salviamo il file nella cartella scriptwork con nome helloworld.sh.

L'estensione .sh è facoltativa ed indica che si tratta di uno script sh/bash. Apriamo una shell e battiamo ./helloworld.sh: non succederà nulla. Questo perchè dobbiamo dare al file il permesso di esecuzione. Battiamo chmod +x helloworld.sh e riproviamo ad eseguirlo con ./: vedremo comparire a schermo la riga “hello world”... Complimenti, avete appena creato un programma!

Le variabili

Di sicuro una delle cose più importanti da sapere per imparare a “scriptare” è conoscere le cosidette “variabili d'ambiente”. Una variabile d'ambiente è una sorta di “condizione” che viene registrata nel registro della shell in esecuzione. Per esempio se nella variabile “fiammifero” inserisco il dato “spento”, con il comando accendi.sh cambierò questa variabile in “acceso”. Lanciando, invece, spegni.sh la variabile “fiammifero” avrà il valore spento.

Vediamo questi due script che lavorano sulla variabile fiammifero.

Accendi.sh

fiammifero=acceso

Spegni.sh

fiammifero=spento

Per vedere qual è il valore del fiammifero, basta lanciare

echo $fiammifero

Le variabili sono molto utili per richiamare situazione passate in altri script.

Il comando read

Ma cosa dobbiamo fare per permettere di registrare una variabile in maniera interattiva? Basta usare il comando read. Vediamo questo script

```
echo “inserisci il tuo nome!!!”

read name

echo “Il tuo nome è $name”
```

Comodo, no?

I simboli di redirezione

Per redirigere l'output di un comando su un file, basta usare l'operatore >. Per esempio per creare un file contenente il nome inserito nella realizzazione 5, basta creare lo script name_txt.sh con la seguente riga di testo

```
echo $name > il_tuo_nome.txt
```

In questo modo, però, il file il_tuo_nome.txt verrà sovrascritto. Per fare in modo che la riga di testo venga aggiunta in coda al file, basta usare >>. Es:

```
echo $name >> il_tuo_nome.txt
```

Script di fine capitolo

Se avete seguito tutte queste relizzazioni, avrete imparato le basi dello scripting. Complimenti!!! Studiamo insieme questo script che riassume tutto ciò imparato.

Clear Cancello la schermata

fiammifero=acceso Imposto la variabile $fiammifero

echo “Benvenuto nello script di fine capitolo”

echo “il fiammifero è $fiammifero”

echo “inserisci il tuo nome!”

read name faccio scrivere all'utente il suo nome e lo riedirigo nella variabile $name

echo “il nome attualmente registrato è $name.”

echo “inserisci il nome del file in cui regitrerò il resoconto dello script!”

read file

echo “grazie! Puoi leggere il resoconto dello script in $file.txt”

echo “Il fiammifero è $fiammifero” > $file.txt Inserisco nel file lo stato del fiammifero

echo “Il tuo nome è $name” >> $file.txt

echo “Ciao ciao!!!”

exit 0 Concludo lo script

Per eseguirlo, copialo in un editor e cancella le spiegazioni!!!

I TRE MOSCHETTIERI DEI FILE DI TESTO: CAT, MORE E LESS

I comandi Cat, More e Less hanno la funzione di visualizzare file di testo semplice. Cominciamo con cat

CAT

Permette di visualizzare i file tutti “in una tirata”. Se il file è molto lungo, esso scorrerà sullo schermo fino alla fine. Una delle comodità di cat, è quella di poter con**cat**enare i file. Per esempio lanciando i comandi

cat file1 file2 > file3

Il file file3 conterrà i due file1 e file2.

MORE

Permette di visualizzare i file schermata per schermata

LESS

Come more, visualizza i file schermata per schermata, ma dispone anche di piccoli strumenti quale motore di ricerca e altro.

Scaricare i file con wget

Soprattutto se stiamo scrivendo un installer, ci capiterà spesso di dover scaricare dei file dalla rete. Niente di più facile:

wget indirizzo

e fa tutto da solo!

Maneggiare i files

È ora arrivato il momento di imparare come cancellare quei migliaia di file creati avendo fatto i giochetti con echo. Impareremo a maneggiare i file

Cominciamo con il più “innocuo”:cp

questo comando, permette di copiare un file cartella/destinazione.

Ecco la sintassi:

cp file_da_copiare cartella_destinazione

Ecco che presentiamo un sistema molto pratico in UNIX: i caratteri jolly.

In parole povere, * significa tutte le cartelle ed i file,*.* tutti i file. Usando questo metodo *.jpg tutte le immagini jpeg e filemio* tutti i file inizianti con filemio: filemio1.txt filemio2.txt filemio3.txt filemio1.html,se voglio copiare solo i filemio.txt,basta eseguire un

cp filemio*.txt cartella di destinazione.

Per copiare una directory,servirà cp -R e per avvisare l'utente prima di sovrascrivere l'eventuale il file, cp -i. gli stessi comandi si possono usare con mv(sposta/rinomina file) o rm(cancella i file senza cestino). Inoltre esiste anche il terribile shred. Esso distrugge ***senza possibilità di recupero*** i file (sovrascrivendoli 20/25 volte). Ora passiamo a dei comandi più innoqui,come mkdir. Esso crea una cartella vuota nella directory indicata. Ad esempio

mkdir ciao

creerà la cartella ciao dentro la cartella corrente. Invece la funzione -p permette di crere delle sottodirectory dentro la cartella creata. In pratica un

mkdir -p ciao/prova/sito

crea la nuova cartella sito dentro la nuova cartella prova e dentro la nuova cartella ciao.

Usare if

If e` un comando presente gia` nei primi linguaggi di programmazione. Se, e` la traduzione italiana di if. Infatti if permette di eseguire un comando se una condizione e` true,vera o un altro comando se la condizione e` false, falsa. La sintassi di if per verificare che una stringa di testo e` uguale ad un`altra e` semplicissimo

if [$cond1==$cond2]

then

comando se stringa uguale

else

comando se stringa diversa

fi

If permette anche di avere altri attriubuti

a nomefile = true se nomefile esiste

b nomefile = true se nomefile e` un dispositivo a blocchi

c nomefile = true se nomefile e` un dispositivo a carattere

d nomefile = true se nomefile e` una directory

e nomefile = true se nomefile esiste

f nomefile = true se nomefile e` un file regolare

g nomefile = true se nomefile ha il set-guid

h nomefile = true se nomefile e` un link simbolico

k nomefile = true se nomefile ha lo sticky bit

r nomefile = true se nomefile e` leggibile

w nomefile = true se nomefile e` scrivibile

x nomefile = true se nomefile e` eseguibile

N nomefile = true se nomefile e` stato modificato l`ultima volta che e` stato letto

file1 -ot file2 = true se file1 e` piu` vecchio di file2 o file2 non esiste

file -ef file2true se file1 e file2 si riferiscono allo stesso device o inode.

stringa1 == stringa2= true se le strinhe sono uguali

stringa1 != stringa2= true se le stringhe non sono ugualeefile e` stato modificato l`ultima volta che e` stato letto

NB: con if sara anche possibile utilizzare espressioni matematiche,come maggiore,minore,uguale,più,meno ecc...

I cicli For

Probabilmente una delle funzioni più comode dei linguaggi di programmazione più "grandi" è la possibilità di istruire il computer per compiere azioni ricursivamente. Anche nello scripting bash questo è possibile. Vediamo.

I cicli for consentono di fare innumerevoli cose.

In questo caso, vedremo come eseguire un comando, tap2wav su una serie di file (con estensione .tap) e poi riedirigere l'output nel nome del file con aggiunta l'estensione .wav:

for n in "*.tap"; do ./tap2wav $n > $n.wav ; done Analizziamo il codice.

Per prima cosa inseriamo nella variabile $n tutti i file con estensione .tap (*.tap) contenuti nella directory corrente.

Poi diciamo al computer di eseguire il comando ./tap2wav $n > $n.wav per ogni file inserito in $n.

Infine, concludiamo il comando dicendo al computer di aver concluso: done.

Vediamo un altro esempio:

for mesi in Gennaio Febbraio Marzo Aprile Maggio Giugno Luglio Agosto Settembre Ottobre Novembre Dicembre do echo $mesi done
In questo caso abbiamo avviato un ciclo che eseguirà per ogni mese inserito il comando echo!

FINE GUIDA NUMERO 1