Automatizzare l'amministrazione di sistema con ssh e scp

by Erdal Mutlu
<erdal(at)linuxfocus.org>

About the author:

Erdal è uno degli editori di LinuxFocus in Turchia. Attualmente sta lavorando come Amministratore di Sistema per la Linotype Library. Essendo stato un grande fan di Linux fin dai tempi dell'Università, gli piace lavorare e sviluppare in questo ambiente.

Abstract:

Se avete un grande numero di sistemi Linux/Unix da amministrare, avrete sicuramente bisogno di alcuni script che aiutino nell'automazione di alcuni processi. Avrete anche notato che nel lavoro di ogni giorno vi trovate a fare spesso le stesse cose su ogni sistema. Forse avrete pensato a un modo di automatizzare questi processi. Questo è vero in particolare per la manutenzione di un grande numero di macchine Linux/Unix configurate allo stesso modo. In questo articolo esporrò un sistema per farlo usando le utilities di ssh.

Introduzione

L'idea è di trovare un modo di copiare alcuni file dalla workstation su cui sto lavorando verso un certo numero di altre workstation o server, e in seguito eseguire alcuni comandi su quelle macchine, come installare qualche rpm o cambiare alcune opzioni di sistema. A volte avremo bisogno di eseguire alcuni comandi su quelle macchine per poi trasferirne i risultati verso la nostra postazione di lavoro.

Per seguire questo articolo avrete bisogno di alcune conoscenze di base della programmazione shell. Per maggiori informazioni sulla programmazione shell potete vedere l'articolo di LinuxFocus sulla Programmazione Shell di Katja e Guido Socher. Dovrete anche conoscere alcune utility di ssh, come ssh−keygen, ssh−add, ssh, scp o sftp. C'è una implementazione libera del protocollo SSH su Linux: OpenSSH, che contiene questi tool. Sono incluse anche le pagine man.

Perchè usare ssh?

La risposta è una domanda: perchè no? Si potrebbe usare rsh−rcp o telnet−ftp, ma non sarebbero la migliore scelta in un ambiente insicuro come Internet e forse anche una Intranet. Ssh fornisce comunicazioni crittate e sicure tra due host attraverso una rete insicura. Non descriverò le implicazioni di sicurezza riguardanti l'uso di questi tool. Date un'occhiata a Attraverso il tunnel, un articolo di Georges Tarbouriech.

A dire il vero, in passato ho usato script basati su telnet/ftp.

Copiare file e directory con scp

Per copiare un file da una directory locale verso un computer remoto si può usare il seguente comando:

scp /path/to/the/file/file1 user@remote_host:/remotedir/newfile

In questo esempio il file con nome file1 viene copiato dalla directory locale all'host remoto (remote_host può essere il nome o l'IP della macchina remota, sotto /remotedir con il nuovo nome newfile. Vi verrà richiesta la password di 'user'. Se l'autenticazione ha successo e l'utente remoto ha i privilegi richiesti, il file viene copiato.
Si può omettere il nome remoto del file. In questo caso il file viene copiato con lo stesso nome.
Semplicemente, significa che si può rinominare un file durante la copia.
Anche l'opposto e` possibile: si può copiare un file remoto sulla macchina locale:

scp user@remote_host:/remotedir/file /path/to/local/folder/newfile

C'è anche un'utile caratteristica di scp. Potete copiare ricorsivamente le directory con l'opzione '−r'.

scp −r user@remote_host:/remotedir .

Questo comando copia la directory 'remotedir' a tutte le sue sotto−directory e file dall'host remoto alla directory corrente con lo stesso nome.

Nota: Si assume che abbiate il daemon sshd attivo sulla macchina remota.

Login remoto con ssh

Al posto di rlogin o telnet si può usare un metodo più sicuro, ssh:

ssh erdal@helvetica.fonts.de

In base alla vostra configurazione, vi verrà richiesto di inserire una password o una frase chiave. Qui ci stiamo connettendo a un computer remoto helvetica.fonts.de con nome di utente remoto erdal. Il comando ssh ha un certo numero di opzioni che possono essere utilizzate in base alle vostre esigenze. Date un'occhiata alla pagina man di ssh.

Eseguire comandi con ssh

C'è la possibilità di eseguire comando sui computer remoti con ssh:

ssh erdal@helvetica.fonts.de df −H

È molto simile alla sintassi per il login remoto. l'unica differenza è dopo il nome di host remoto. Il comando (in questo caso 'df −H') viene passato per l'esecuzione alla macchina remota. L'output del comando viene mostrato sul terminale.

Connettersi a un computer remoto senza password

Invece di usare l'autenticazione tramite password, si può usare una coppia di chiavi (pubblica/privata). Dovete generare la vostra coppia di chiavi. C'è il comando ssh−keygen che può essere utilizzato per generare chiavi per ssh:

ssh−keygen −b 1024 −t dsa

Vi verrà richiesto un nome per la chiave privata. Normalmente il nome della chiave pubblica è lo stesso della chiave privata con estensione '.pub'. Qui '−b 1024' è il numero di bit nella chiave da creare. Se non lo specificate verrà usato il valore di default. La '−t dsa' viene usata per specificare il tipo di chiave. I valori possibili sono 'rsa1' per il protocollo versione 1 e 'rsa' o 'dsa' per il protocollo versione 2. Io raccomando l'uso della versione 2 del protocollo SSH. Ma se avete dei server vecchi che supportano solo la versione 1, dovrete specificare '−t rsa1' e creare un altro paio di chiavi. Potete forzare ssh a usare il protocollo versione 1 o il protocollo versione 2 specificando rispettivamente '−1' o '−2'.

Per poter usare le vostre chiavi dovete installare la vostra chiave pubblica sulla macchina remota. Il contenuto del file della chiave pubblica dovrebbe essere copiato o accodato a $HOME/.ssh/authorized_keys o a $HOME/.ssh/authorized_keys2 . Fate attenzione a non mischiare le chiavi per i diversi protocolli. authorized_keys viene usato per il protocollo versione 1. authorized_keys2 viene usato per la versione 2. Se avete installato correttamente la vostra chiave pubblica, la prossima volta che vi collegherete a quella macchina vi verrà richiesto di inserire la vostra password e, se la sbagliate, vi verrà chiesta la password del sistema remoto. Potete limitare l'accesso ai vostri sistemi al solo uso della chiave pubblica modificando il file di configurazione di sshd. Il nome del file è /etc/ssh/sshd_config e il parametro da modificare è 'PasswordAuthentication'. Cambiate il valore di questo parametro a no (PasswordAuthentication no) e fate ripartire sshd.

Fino a qui tutto a posto. Abbiamo un modo sicuro di copiare ed eseguire comandi su un sistema remoto. Ma per l'automazione di alcuni lavori non dovremmo digitare password. Altrimenti non potremmo automatizzare niente. Una soluzione potrebbe essere quella di scrivere la password in ogni script, che comunque non è una buona idea. La cosa migliore è di usare il key−agent per prendersi cura delle nostre password. ssh−agent è un programma che tiene le chiavi private usate per l'autenticazione con chiave pubblica. Dovreste lanciare un key−agent:

ssh−agent $BASH

e aggiungere le vostre chiavi private usando

ssh−add .ssh/id_dsa

o

ssh−add .ssh/identity

id_dsa è il file di chiavi private DSA e identity è il file delle chiavi private RSA1. Questi sono i nomi di default dati durante la generazione delle chiavi con ssh−keygen. Naturalmente vi verrà chiesta la vostra password prima che ssh−add aggiunga le vostre chiavi a ssh−agent. Potete vedere quali chiavi sono state aggiunte con il seguente comando:

ssh−add −l

Ora, se vi connettete al server che ha la vostra chiave nel file delle autorizzazioni, verrete connesis senza dover scrivere niente! ssh−agent avrà cura di completare il processo di autenticazione.

Quando usate ssh−agent come descritto sopra, potete usarlo solo dal terminale da cui ssh−agent è stato lanciato. Se volete usare ssh−agent da tutti i terminali che aprite, c'è bisogno di un po' piu` di lavoro,. Ho scritto questo piccolo script per avviare l'agent:

```
#!/bin/sh
#
# Erdal mutlu
#
# Starting an ssh-agent for batch jobs usage.

agent_info_file=~/.ssh/agent_info

if [ -f $agent_info_file ]; then
 echo "Agent info file : $agent_info_file exists."
 echo "make sure that no ssh-agent is running and then delete this file."
 exit 1
fi

ssh-agent | head -2 > $agent_info_file
chmod 600 $agent_info_file
exit 0
```

Questo script controlla l'esistenza di un file chiamato agent_info nella home dell'utente dove vengono normalmente tenuti i file di ssh riguardanti l'utente. Nel nostro caso la directory è '.ssh/'. Se il file esiste l'utente viene avvisato della presenza del file e riceve un breve messaggio su ciò che può fare. Se l'utente non sta già usando ssh−agent deve cancellare il file e far ripartire lo script. Lo script lancia ssh−agent e salva le prime due righe di output nel file agent_info. Queste informazioni verranno poi usate dalle utility ssh. Quindi c'è una linea che cambia i permessi del file, in modo che solo il proprietario del file possa leggerci e scriverci.

Una volta che l'agent è in funzione potete aggiungervi chiavi. Ma prima di farlo dovete usare source sul file agent_info, in modo che i tool ssh sappiano dove trovare l'agent:

source ~/.ssh/agent_info or . ~/.ssh/agent_info

E aggiungete le vostre chiavi con ssh−add. Potete aggiungere le righe seguenti al vostro .bashrc, in modo da lanciare source su agent_info ogni volta che aprite un terminale:

```
if [ -f .ssh/agent_info ]; then
. .ssh/agent_info
fi
```

ATTENZIONE: Dovete avere la sicurezza dell'host da cui usate ssh−agent e gli script automatizzati che sto per descrivervi. Altrimenti, chiunque abbia accesso al vostro account potrà avere accesso a tutti i server a cui potete accedere tramite chiavi ssh. Ogni cosa ha il suo prezzo!

Lo script

È venuto il momento di spiegare come andremo ad automatizzare alcuni lavori da amministratore di sistema. L'idea è di eseguire un insieme di comandi per una lista di host e di prendere o mandare alcuni file da e per tali host. È una cosa che gli amministratori di sistema devono fare spesso. Ecco lo script:

```
#!/bin/sh

# Installing anything using Secure SHELL and SSH agent
# Erdal MUTLU
# 11.03.2001


#####################################################################
#                       Functions                                   #
#####################################################################
### Copy files between hosts
copy_files()
{
 if [ $files_file != "files_empty.txt" ];then
  cat $files_file | grep -v "#" | while read -r line
  do
   direction=`echo ${line} | cut -d " " -f 1`
   file1=`echo ${line}     | cut -d " " -f 2`
   file2=`echo ${line}     | cut -d " " -f 3`

   case ${direction} in
      "l2r") :  ### From localhost to remote host
          echo "$file1  --> ${host}:${file2}"
          scp $file1 root@${host}:${file2}
      ;;
      "r2l") :  ### From remote host to localhost
          echo "${host}:${file2}  --> localhost:${file2}"
          scp root@${host}:${file1} ${file2}
      ;;
      *)
          echo "Unknown direction of copy : ${direction}"
          echo "Must be either local or remote."
      ;;
   esac
  done
 fi
}

### Execute commands on remote hosts
execute_commands()
{
 if [ $commands_file != "commands_empty.txt" ];then
  cat $commands_file | grep -v "#" |  while read -r line
  do
   command_str="${line}"
   echo "Executing $command_str ..."
   ssh -x -a root@${host}  ${command_str} &
   wait $!
   echo "Execute $command_str OK."
  done
 fi
}

### Wrapper function to execute_commands and copy_files functions
doit()
{
```

```
 cat $host_file | grep -v "#" | while read -r host
 do
  echo "host=$host processing..."
  case "${mode}" in
        "1")
                copy_files
                execute_commands
                ;;
        "2")
                execute_commands
                copy_files
                ;;
        *)
                echo "$0 : Unknown mode : ${mode}"
                ;;
  esac
  echo "host=$host ok."
  echo "-----------------------------------------------------------------"
 done
}

######################################################################
### Program starts here
######################################################################

if [ $# -ne 4 ]; then
 echo "Usage : $0 mode host_file files_file commands_file"
 echo ""
 echo "mode is 1 or 2 "
 echo "    1 : first copy files and then execute commands."
 echo "    2 : first execute commands and then copy files."
 echo "If the name of files.txt is files_empty.txt then it is not processed."
 echo "If the name of commands.txt is commands_empty.txt then it is
 echo "not processed."
 exit
fi


mode=$1
host_file=$2
files_file=$3
commands_file=$4

agent_info_file=~/.ssh/agent_info
if [ -f $agent_info_file ]; then
 . $agent_info_file
fi

if [ ! -f $host_file ]; then
 echo "Hosts file : $host_file does not exist!"
 exit 1
fi

if [ $files_file != "files_empty.txt" -a ! -f $files_file ]; then
  echo  "Files file : $files_file does not exist!"
  exit 1
fi

if [ $commands_file != "commands_empty.txt" -a ! -f $commands_file ]; then
 echo  "Commands file : $commands_file does not exist!"
 exit 1
fi

#### Do everything there
```

```
doit
```

Salviamo lo script come ainstall.sh (automated installation, installazione automatizzata) e proviamo a lanciarlo senza parametri. Riceveremo il seguente messaggio:

./ainstall.sh

```
Usage : ./ainstall.sh mode host_file files_file commands_file

mode is 1 or 2
        1 : first copy files and then execute commands.
        2 : first execute commands and then copy files.
If the name of files.txt is files_empty.txt then it is not processed.
If the name of commands.txt is commands_empty.txt then it is not
processed.
```

Come dice il messaggio, se non volete eseguire nessun comando, passate commands_empty.txt come nome del file di comandi; se non volete trasferire alcun file, passate files_empty.txt come nome del file con la lista dei file da trasferire. A volte dovete solo eseguire qualche comando, mentre altre volte avete bisogno di trasferire qualche file.

Prima di spiegare lo script linea per linea lasciatemi fare un esempio di uso: Supponiamo che abbiate aggiunto un DNS secondario alla vostra rete e che vogliate aggiungerlo al file /etc/resolf.conf . Per semplictà supponiamo che tutti i vostri host abbiano lo stesso resolv.conf quindi l'unica cosa che dovete fare è di copiare il nuovo resolv.conf in tutti gli host.
Prima di tutto avete bisogno di una lista degli host. La scriveremo in un file chiamato hosts.txt. Il formato del file hosts.txt è di un hostname o un IP per linea. Ecco un esempio:

```
##############################################################################
#### Every line contains one hostname or IP address of a host. Lines that
#### begin with or contain # character are ignored.
##############################################################################
helvetica.fonts.de
optima.fonts.de
zaphino
vectora
#10.10.10.162
10.10.10.106
193.103.125.43
10.53.103.120
```

Come potete vedere dall'esempio, si possono specificare hostname completi o solo il nome host. Quindi avete bisogno di un file dove elencare i file da trasferire. Ci sono due possibilità di trasferimento:

- Da locale a tutti gli host elencati in hosts.txt. Questo è il nostro caso.
- Da ogni host elencato in hosts.txt a localhost. Questo serve quando preleviamo qualche file da ogni host. Per esempio l'uso col nostro piccolo script di backup che vedremo più avanti nell'articolo.

I file da trasferire sono elencati in un altro file. Salviamolo come files_file.txt. Il formato è il seguente: ogni linea include informazioni su come copiare un singolo file. Ci sono due possibili direzioni di copia: l2r (locale a remoto) e r2l (remoto a locale). l2r si usa quando un file viene copiato dal computer locale a quelli remoti. r2l quando un file viene copiato da un computer remoto a quello locale. Dopo la parola chiave per la direzione ci sono due nomi di file. I campi sono separati da spazi o tabulatori. Il primo file viene copiato sul secondo in base alla direzione indicata. Il nome file per l'host remoto deve essere completo, altrimenti verrà copiato nella home dell'utente root. Questo è il nostro file files_file.txt

```
##############################################################################
# The structure of this file is :
```

```
# - The meaning of the fileds are : is l2r (localhost to remote) and r2l
# (remote computer to local).
#       r2l  file1   file2
#         means copy file1 from remote (hosts specified in the
#         hosts.txt file) computer to localhost as file2.
#       l2r     file1   file2
#         means copy file1 from localhost to
#         remote (hosts specified in the hosts.txt file) computer as file2
#         file1 and file2 are files on the corrsponding hosts.
#
#       Note: the order of using local and remote specifies the direction
#       of the copy process.
##############################################################################
l2r     resolv.conf     /etc/resolv.conf
```

Come vedete ho già incluso la descrizione su come il file è strutturato. Normalmente includo questa descrizione per ogni files_file.txt che uso. È una soluzione semplice ma buona per tenere documentazione. Nel nostro esempio vogliamo copiare resolv.conf su un host remoto come /etc/resolv.conf. Per fare una dimostrazione, dopo averlo copiato sugli host remoti ho incluso dei comandi per cambiare proprietario e gruppo e per mostrare i contenuti di /etc/resolv.conf. I comandi da eseguire vengono messi in un file separato. Chiamiamo questo file commands_file.txt. Eccolo qui:

```
##############################################################################
# The structure of this file is : Every line contains a command to be
#  executed. Every command is treated seperately.
##############################################################################
chown root.root /etc/resolv.conf
chmod 644 /etc/resolv.conf
cat /etc/resolv.conf
```

Il file dei comando contiene le istruzioni che dovranno essere eseguite su ogni host elencato in hosts.txt. I comandi vengono eseguiti in ordine sequenziale, cosi` come sono elencati nel file.

OK, ora avete tutti i file necessari per l'esempio. L'unica cosa che manca è l'opzione 'mode' che indica quale dei due tra commands_file.txt e files_file.txt deva essere eseguito per primo. Si può prima trasferire i file e poi eseguire i comandi con mode=1. L'opposto, prima eseguire i comandi e poi trasferire i files, con mode=2. Ora potete eseguire lo script con gli argomenti richiesti in questo modo:

./ainstall.sh 1 hosts.txt files_file.txt commands_file.txt

Un piccolo trucco: normalmente io metto come prefisso a files.txt la stringa files_ e di seguito un nome descrittivo, come files_resolvconf.txt. La stessa tecnica la applico a hosts.txt ecommands.txt.

Ora è il momento di spiegare un po' come funziona lo script. Il programma inizia controllando il numero di argomenti passati e, se diverso da 4, mostra il messaggio di istruzioni d'uso. Se il numero di argomenti è esatto, gli argomenti vengono assegnati alle rispettive variabili. Quindi, se '~/.ssh/agent_info' esiste viene caricato (con source). Questo file contiene informazioni sull'agent ssh in esecuzione. Se non usate un agent dovrete inserire le password manualmente, il che significa che non c'è automazione :). Dopodichè di ogni file (hosts, files e commands) viene verificata l'esistenza. C'è anche un test speciale per files_empty.txt e commands_empty.txt. Se avete specificato uno di questi nomi, non c'è bisogno del controllo. Ho cambiato questa part dello script durante la stesura di questo articolo. Prima era solo:

```
if [ -f $host_file -a -f $files_file -a -f $commands_file ]; then
 echo "$host_file $files_file $commands_file"
 doit
else
 echo "$host_file or $files_file or $commands_file does not exist"
 exit
fi
```

In questo caso dovevo avere dei file con nome files_empty.txt e commands_empty.txt. Ma non era un problema perchè lavoravo sempre nella stessa directory.
Infine c'è la chiamata alla funzione 'doit'. Questa funzione controlla tutto. La funzione contiene un loop con 'cat' e 'while' che, per ogni host del file '$hosts_file' richiama copy_files e execute_commands in base alle impostazioni di 'mode'. Quindi il lavoro viene eseguito per ogni host. La variabile 'host' contiene l'hostname o l'IP attuale.
Diamo un'occhiata alla funzione copy_files. Questa prima controlla se il valore di 'files_file' corrisponde a 'files_empty.txt' o no. Se corrisponde non viene eseguito nulla. Altrimenti per ogni linea di '$files_file' le variabili 'direction', 'file1' e 'file2' contengono la direzione di copia, il primo e il secondo nome di file rispettivamente. In base alla 'direction' viene effettuata una copia con scp.
Infine diamo un'occhiata a cosa succede in execute_commands. La funzione controlla se la variabile 'commands_file' contiene 'commands_empty.txt' o meno. Se corrisponde non viene eseguito nulla. Altrimenti ogni comando di '$commands_file' viene eseguito sull'host remoto usando ssh in background. Dopo l'esecuzione di ssh c'è una chiamata a wait con parametro '$!'. Questo comando si assicura che i comandi vengano eseguiti uno dopo l'altro. '$!' viene espanso all'ID di processo del comando eseguito in background più recente.
Tutto qui. Semplice, no?

Un semplice backup dei vostri file di configurazione

Questo è un uso un po' più avanzato dello script. L'idea è di lanciare il backup dei file di configurazione dei vostri host o server. Per questo scopo ho scritto un piccolo script che usa ainstall.sh:

```
#!/bin/sh

server_dir=${HOME}/erdal/sh/ServerBackups

if [ ! -d $server_dir ]; then
 echo "Directory : $server_dir does not exists."
 exit 1
fi

cd $server_dir

servers=ll_servers.txt
prog=${HOME}/erdal/sh/einstall_sa.sh

cat $servers | grep -v "#" | while read -r host
do
  echo $host > host.txt
  $prog 1 host.txt files_empty.txt
  servers/${host}/commands_make_backup.txt
  $prog 1 host.txt files_getbackup.txt commands_empty.txt
  mv -f backup.tgz servers/${host}/backup/`date +%Y%m%d`.tgz
  rm -f host.txt
done

exit 0
```

Dovete avere una directory speciale chiamata 'servers'. Sotto questa directory ci devono essere due file: files_getbackup.txt e ll_servers.txt. Questo è il file 'files_getbackup.txt':

r2l /root/backup.tgz backup.tgz

'll_servers.txt' contiene i nomi o gli IP degli host di cui fare il backup. Ogni hostname elencato in 'll_servers.txt' deve avere una directory con lo stesso nome e sotto questa directory deve esserci un file chiamato commands_make_backups.txt, che contiene un comando per creare un file /root/backup.tgz con tutti i file di configurazione di quell'host. E una directory di nome 'backup'. Tutti i backup di questo host verranno archiviati in questa directory. Se il contenuto di ll_servers.txt è:

```
fileserver
dbserver
10.10.10.1
appserver
```

La struttura di directory della vostra directory '$servers' deve essere così:

```
servers
|-- files_getbackup.txt
|-- ll_servers.txt
|-- make_server_backups.sh
|-- 10.10.10.1
|   |-- backup
|   `-- commands_make_backup.txt
|-- appserver
|   |-- backup
|   `-- commands_make_backup.txt
|-- dbserver
|   |-- backup
|   `-- commands_make_backup.txt
|-- fileserver
    |-- backup
    `-- commands_make_backup.txt
```

e questi sono alcuni esempi di comandi per i file commands_make_backups.txt :

tar cfz /root/backup.tgz /etc/samba /etc/atalk /etc/named.conf /var/named/zones

che serve per fare backup di samba, atalk e della configurazione del DNS e delle sue zone.

tar cfz /root/backup.tgz /etc/httpd /usr/local/apache

che serve per fare backup della configurazione di apache e dei suoi file.

tar cfz /root/backup.tgz /etc/squid /etc/named.conf

che serve per fare backup della configurazione di squid e del DNS secondario.

Usando lo script qui sopra e costruendo i file commands_make_backup.txt secondo le vostre esigenze potete effettuare i backup delle configurazioni dei vostri server.

Conclusione

Lo script ainstall.sh consente di automatizzare alcuni lavori di amministrazione di sistema. Lo script è basato su un uso semplice dei tool di ssh. Apprezzerete questo script in presenza di un grande numero di sistemi identici.

Riferimenti

- SSH, The Secure Shell: The Definitive Guide, by Daniel J. Barrett and Richard Silverman.
- Attraverso il tunnel, by Georges Tarbouriech.
- Programmazione Shell, by Katja and Guido Socher.

2005−01−10, generated by lfparser_pdf version 2.51