Automatisierung der Systemadministration mittels ssh und scp

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

About the author:

Erdal ist einer der türkischen LF–Editoren. Derzeit arbeitet er als System–Administrator für Linotype Library. Da er seit seiner Universitätszeit ein großer Linux–Fan ist, liebt er es, in dieser Umgebung zu arbeiten und zu entwickeln.

Abstract:

Wenn Sie eine große Anzahl von Linux–/Unix–Systemen administrieren müssen, benötigen Sie sicherlich einige Skripte, die Ihnen bei der Automatisierung einiger Arbeitsabläufe helfen. Sie werden während Ihrer täglichen Arbeit bemerkt haben, dass Sie auf allen Systemen ähnliche Arbeiten verrichten. Vielleicht haben Sie schon daran gedacht, einige dieser Prozesse zu automatisieren. Dies ist besonders sinnvoll für die Verwaltung einer großen Anzahl von Linux–/Unix–Systemen, die identisch konfiguriert sind. In diesem Artikel werde ich einen Weg vorstellen, wie man dies mit Hilfe der ssh–Programme machen kann.

Einführung

Es geht darum, einen Weg zu finden, um einige Dateien von dem Rechner, vor dem ich sitze, zu einer Anzahl von anderen Rechnern zu kopieren und danach einige Befehle auf diesen Maschinen wie z. B. rpm–Installationen oder Ändern einiger Systemoptionen auszuführen. Manchmal wird es erforderlich sein, zuerst einige Befehle auf diesen Maschinen auszuführen und danach einige Dateien abzurufen, die das Ergebnis der ausgeführten Befehle sein könnten.

Um diesem Artikel folgen zu können, benötigen Sie grundlegende Kenntnisse über Shell–Programmierung. Wegen weiterer Informationen zur Shell–Programmierung schauen Sie sich den Artikel Shell Programmierung von Katja und Guido Socher an. Sie benötigen auch Wissen über die ssh–Programme wie ssh–keygen, ssh–add, ssh, scp oder sftp. Es gibt eine freie Implementation des SSH–Protokolls unter Linux OpenSSH, die alle diese Programme enthält. Außerdem gibt es auch Handbuch–Seiten dazu.

Warum sollte man ssh benutzen?

Die Antwort ist eine Frage: Warum nicht? Man könnte rsh−rcp oder telnet−ftp benutzen, sie sind in unsicheren Umgebungen wie dem Internet und vielleicht auch dem Intranet nicht angemessen. Ssh bietet sichere verschlüsselte Kommunikation zwischen zwei Rechnern über ein unsicheres Netzwerk. Ich werde hier nicht über Sicherheits−Implikationen sprechen, wenn man diese Programme benutzt. Schauen Sie sich dazu den Artikel Durch den Tunnel von Georges Tarbouriech an.
Auch ich habe in der Vergangenheit Skripte benutzt, die auf telnet/ftp basierten.

Kopieren von Dateien und Verzeichnissen mittels scp

Zum Kopieren einer Datei aus einem lokalen Verzeichnis auf einen entfernten Rechner kann folgender Befehl benutzt werden:

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

In diesem Beispiel wird die Datei file1 aus dem lokalen Ordner auf den entfernten Rechner (dies kann die IP−Adresse oder der Rechnername sein) in das Verzeichnis /remotedir unter dem Namen newfile kopiert. Sie werden aufgefordert, sich als 'user' zu authentifizieren. Wenn die Authentifizierung erfolgreich ist und der entfernte Benutzer die entsprechenden Rechte besitzt, wird die Datei kopiert. Man kann den Ziel−Dateinamen auslassen, dann wird die Datei mit dem gleichen Namen kopiert. Kurz gesagt, man kann Dateien während des Kopierens umbennen.
Das Gegenteil ist auch möglich: Sie können eine entfernte Datei in einen lokalen Ordner kopieren.

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

Der scp−Befehl hat noch eine sehr nette Eigenschaft: Sie können Verzeichnisse mit der '−r'−Option rekursiv kopieren.

scp −r user@remote_host:/remotedir .

Der obige Befehl kopiert das Verzeichnis 'remotedir' und alle darunterliegenden Verzeichnisse und Dateien vom entfernten Rechner in das aktuelle Verzeichnis mit dem gleichen Namen.

Hinweis: Dabei wird angenommen, dass Sie auf dem entfernten Rechner den sshd−Dämon gestartet haben.

Entfernte Anmeldung mittels ssh

Anstelle von rlogin oder telnet kann man einen sichereren Weg wählen, nämlich ssh:

ssh erdal@helvetica.fonts.de

Abhängig von Ihrer Konfiguration werden Sie entweder nach Ihrem Passwort oder Ihrer Passphrase gefragt. Hier verbinden wir uns mit dem entfernten Rechner helvetica.fonts.de und der dortigen Benutzerkennung erdal. Der ssh−Befehl bietet eine Anzahl von Optionen, die Sie entsprechend Ihren Bedürfnissen nutzen können. Schauen Sie sich die Handbuchseite zu ssh an.

Befehlsausführung mittels ssh

Es gibt die Möglichkeit, mittels ssh Befehle auf dem entfernten Rechner auszuführen:

ssh erdal@helvetica.fonts.de df −H

Die Syntax entspricht sehr dem remote−login−Befehl. Der einzige Unterschied ist der Teil nach dem Rechnernamen. Der Befehl (in diesem Beispiel 'df −H') wird zur Ausführung an den entfernten Rechner übergeben. Die Ausgabe des Befehls wird auf Ihrem Terminal angezeigt.

Verbindung zu einem entfernten Rechner ohne Passwort

Anstelle der Passwort−Authentifizierung können Sie ein Schlüsselpaar (öffentlich/privat) nutzen. Sie müssen Ihr Schlüsselpaar generieren. Dafür gibt es das Programm ssh−keygen, das zum Erzeugen der Schlüssel für ssh genutzt werden kann:

ssh−keygen −b 1024 −t dsa

Sie werden nach dem Namen Ihres privaten Schlüssels gefragt. Normalerweise ist der Name des öffentlichen Schlüssels gleich dem privaten Schlüssel, jedoch um '.pub' ergänzt. '−b 1024' bezeichnet hier die Anzahl der Bits in dem zu erstellenden Schlüssel. Wenn Sie diesen Wert nicht spezifizieren, wird ein Fehlwert eingesetzt. '−t dsa' dient zur Angabe des Schlüsseltyps. Die möglichen Werte sind 'rsa1' für Protokollversion 1 und 'rsa' oder 'dsa' für Protokollversion 2. Ich empfehle, Protokollversion 2 zu benutzen. Aber wenn Sie ältere Server haben, die nur Version 1 unterstützen, müssen Sie '−t rsa1' angeben und ein weiteres Schlüsselpaar erzeugen. Sie können ssh dazu zwingen, Protokollversion 1 oder Protokollversion 2 zu nutzen, indem Sie entweder '−1' oder '−2' angeben.

Um Ihren Schlüssel zu nutzen, sollten Sie Ihren öffentlichen Schlüssel auf dem entfernten Rechner installieren. Der Inhalt der Datei mit dem öffentlichen Schlüssel sollte kopiert oder angehängt werden auf die Dateien $HOME/.ssh/authorized_keys oder $HOME/.ssh/authorized_keys2. Seien Sie vorsichtig und mischen Sie keine Schlüssel für verschiedene Protokollversionen. Für Protokollversion 1 wird authorized_keys und für Protokollversion 2 authorized_keys2 genutzt. Wenn Sie Ihren öffentlichen Schlüssel korrekt installiert haben, werden Sie bei der nächsten Verbindung zu diesem Computer zunächst nach Ihrer Passphrase gefragt und, wenn dies fehlschlägt, nach dem Passwort der entfernten Benutzerin. Sie können die Verbindungen zu Ihren Systemen auf Authentifizierung mit öffentlichen Schlüsseln beschränken, indem Sie die sshd−Konfigurationsdatei editieren. Der Dateiname ist /etc/ssh/sshd_config und der entsprechende Parameter ist 'PasswordAuthentification'. Ändern Sie diesen Parameterwert auf no (PasswordAuthentification no) und starten Sie sshd neu.

Bis zu diesem Punkt ist alles okay. Wir haben einen sicheren Weg zum Kopieren und Ausführen von Befehlen auf entfernten Systemen. Aber zum Automatisieren einiger Jobs sollten wir keine Passwörter oder Passphrasen eingeben müssen. Sonst können wir nichts automatisieren. Eine Lösung könnte sein, die erforderlichen Angaben in jedem Skript einzutragen, was auf keinen Fall eine gute Idee ist. Der bessere Weg ist es, den Schlüsselagenten zur Behandlung unserer Passphrasen zu nutzen. Ssh−agent ist ein Programm, um für Authentifizierung durch öffentliche Schlüssel benutzte private Schlüssel aufzunehmen. Sie sollten einen Schlüsselagenten starten:

ssh−agent $BASH

und ihm Ihre privaten Schlüssel hinzufügen mittels

ssh−add .ssh/id_dsa

oder

ssh−add .ssh/identity

Id_dsa ist die DSA−Datei mit privaten Schlüsseln und identity die RSA1−Datei mit privaten Schlüsseln. Dies sind die Standard−Dateinamen, die während der Schlüssel−Erstellung mittels ssh−keygen verwandt werden. Natürlich werden Sie nach Ihrer Passphrase gefragt, bevor ssh−add Ihren Schlüssel dem ssh−agent hinzufügt. Sie können mit dem folgenden Befehl auflisten, welche Schlüssel hinzugefügt wurden:

ssh−add −l

Wenn Sie sich nun mit einem entfernten Rechner verbinden, der Ihren Schlüssel in der autorisierten Datei enthält, werden Sie verbunden, ohne irgendetwas eingeben zu müssen! Der ssh−agent kümmert sich um den Authentifizierungsprozess.

Wenn Sie ssh−agent wie oben beschrieben nutzen, können Sie es nur in dem Terminal benutzen, indem es gestartet wurde. Wenn Sie ssh−agent von jedem Terminal nutzen möchten, das Sie öffnen, müssen Sie etwas mehr tun. Ich habe das folgende kleine Skript zum Starten des Agenten geschrieben:

```
#!/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
```

Das Skript testet auf das Vorhandensein einer Datei namens agent_info im Start−Verzeichnis der Benutzerin, wo sich normalerweise ssh−bezogene Dateien finden. In unserem Fall ist dies das Verzeichnis '.ssh'. Wenn die Datei existiert, wird der Benutzer wegen des Vorhandenseins der Datei gewarnt und er erhält eine kleine Nachricht dazu, was getan werden kann. Wenn ssh−agent nicht schon läuft, muss die Datei gelöscht werden und das Skript erneut gestartet werden. Das Skript startet ssh−agent und speichert die zwei ersten Zeilen der Ausgabe in die Datei agent_info. Diese Information wird von den anderen ssh−Programmen benutzt. Die nächste Zeile ändert den Modus der Datei, so dass nur die Eigentümerin der Datei diese lesen und beschreiben kann.

Wenn Ihr Agent läuft, können Sie ihm Ihre Schlüssel hinzufügen. Vorher müssen Sie jedoch die agent_info−Datei in Ihre aktuelle Umgebung einfügen (sourcen), damit die ssh−Programme wissen, wo sich Ihr Agent befindet:

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

Und fügen Sie dann Ihre Schlüssel mittels ssh−add hinzu. Sie können die folgenden Zeilen Ihrer .bashrc−Datei hinzufügen, so dass bei jeder neuen Terminal−Sitzung die Datei agent_info eingelesen wird:

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

WARNUNG: Sie müssen den Rechner sichern, von dem aus Sie ssh−agent und das hier beschriebene automatisierte Skript nutzen. Wenn ansonsten jemand Zugang auf Ihr Konto hat, kann diese Person auf alle Server zugreifen, die Sie mit Ihren ssh−Schlüsseln nutzen. Alles hat seinen Preis!

Das Skript

Nun ist es an der Zeit zu erklären, wie wir einige Jobs eines System−Administrators automatisieren werden. Wir wollen eine Anzahl von Befehlen für eine gegebene Liste von Rechnern ausführen und einige Dateien von/zu diesen Rechnern abrufen/übertragen. Das sind häufige Aufgaben einer Systemadministratorin. Hier ist das Skript:

```
#!/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
```

Wir speichern dieses Skript als aiinstall.sh (automatisierte Installation) und starten es ohne Parameter. Wir erhalten die folgende Nachricht:

./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.
```

Wie die Meldung besagt: Wenn Sie keine Befehle ausführen wollen, geben Sie commands_empty.txt als Argument commands_file an und wenn Sie keine Dateien übertragen wollen, dann benutzen Sie files_empty.txt als Ersatz für das Argument files_file. Manchmal müssen Sie nur einige Befehle ausführen, ein anderes Mal nur Dateien übertragen.

Bevor wir das Skript Zeile für Zeile erklären, will ich eine Beispiel−Anwendung geben: Nehmen Sie an, dass Sie einen sekundären DNS−Server zu Ihrem Netzwerk hinzugefügt haben und Sie diesen zur /etc/resolv.conf−Datei hinzufügen müssen. Der Einfachheit halber nehmen wir an, dass alle Ihre Rechner die gleiche resolv.conf−Datei verwenden. Das einzige, was Sie also tun müssen, ist, die neue resolv.conf−Datei auf alle Rechner zu kopieren.
Zuerst brauchen Sie eine Liste Ihrer Rechner. Wir schreiben alle Rechner in eine Datei namens hosts.txt. Das Format dieser Datei ist so, dass jede Zeile nur einen Hostnamen oder eine IP−Adresse enthält. Hier ist ein Beispiel:

```
#############################################################################
#### 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
```

Wie Sie aus dem Beispiel sehen können, können Sie voll qualifizierte Hostnamen oder nur den Rechnernamen eingeben. Dann benötigen Sie eine Datei, in der Sie die zu übertragenden Dateien angeben. Es gibt zwei mögliche Transfer−Typen:

- Vom lokalen Rechner zu den in der hosts.txt aufgeführten Rechnern. Dies ist unser Fall.
- Von jedem der in der hosts.txt aufgeführten Rechner zum lokalen Rechner. Das ist der Fall, wenn wir einige Dateien von jedem Rechner abrufen. Zum Beispiel in unserem einfachen Sicherungsskript, das ich später in diesem Artikel beschreiben werde.

Die zu übertragenden Dateien werden in einer weiteren Datei aufgelistet. Diese speichern wir als files_file.txt. Das Format der files_file.txt−Datei ist wie folgt: Jede Zeile enthält Informationen zum Kopieren jeweils einer Datei. Es gibt zwei mögliche Kopierrichtungen: l2r (lokal to remote) und r2l (remote to lokal). L2r ist der Fall, wenn eine Datei vom lokalen Rechner zu einem entfernten Rechner kopiert wird. R2l bedeutet das Kopieren einer Datei vom entfernten Rechner zum lokalen Rechner. Felder werden durch Leerzeichen oder Tabs getrennt. Die erste Datei wird entsprechend dem Richtungsparameter zur zweiten kopiert. Der Name der Datei auf dem entfernten Rechners sollte den vollständigen Pfad enthalten, da sie sonst in das Startverzeichnis des Benutzers root kopiert wird. Hier ist unsere Datei 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
```

Wie Sie sehen, habe ich bereits eine Beschreibung der Dateistruktur eingefügt. Normalerweise mache ich das für jede files_file−Datei, die ich benutze. Es ist eine einfache, aber gute Lösung für die Dokumentation. In unserem Beispiel wollen wir die resolv.conf−Datei als /etc/resolv.conf auf entfernte Rechner kopieren. Zu Demonstrationszwecken habe ich nach dem Kopiervorgang einige Befehle hinzugefügt, um Eigentümer und Berechtigungen der Datei zu ändern und deren Inhalt anzuzeigen. Auszuführende Befehle werden in einer separaten Datei eingetragen. Wir nennen unsere Befehlsdatei commands_file.txt. Hier ist ihr Inhalt:

```
##############################################################################
# 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
```

Die Befehlsdatei enthält Befehle, die auf jedem der in der hosts.txt−Datei aufgeführten Rechner ausgeführt werden. Befehle werden in einer sequentiellen Reihenfolge abgearbeitet, d. h. chown ist der erste ausgeführte Befehl und danach kommt der nächste.

Okay, nun haben wir alle Dateien, die wir für dieses einfache Beispiel benötigen. Das einzige, was noch spezifiert werden muss, ist die 'mode'−Option, die angibt, welche der beiden Dateien commands_file.txt oder files_file.txt zuerst verarbeitet werden muss. Man kann die in der files_file−Datei aufgeführten Dateien

kopieren und dann alle Befehle auf den Zielrechnern ausführen, das ist Modus 1. Und das Gegenteil, Ausführen von Programmen und danach Dateiübertragung, entspricht 'mode=2'. Nun können Sie das Skript mit den benötigten Argumenten wie folgt ausführen:

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

Ein kleiner Tip: Normalerweise benutze ich für files.txt den Prefix files_ und dann einen kurzen beschreibenden Namen, wie files_resolv.conf.txt. Die gleiche Technik benutze ich für hosts.txt und commands.txt.

Nun ist es an der Zeit, das Skript selbst etwas zu erläutern. Das Programm überprüft zunächst die Anzahl der Argumente und wenn diese nicht 4 ist, erscheint der Benutzungshinweis. Wenn die Argumentanzahl richtig ist, werden die Argumente entsprechenden Variablen zugewiesen. Als nächstes wird '~/.ssh/agent_info' eingelesen, wenn diese existiert. Diese Datei enthält Informationen über Ihren aktiven ssh−Agenten. Wenn Sie keinen Agenten nutzen, müssen Sie Passwörter oder Passphrasen manuell eingeben, d. h. aber keine Automation:). Danach wird jede Parameterdatei (Rechner, Dateien und Befehle) auf Vorhandensein getestet. Es gibt auch einen speziellen Test für files_empty.txt und commands_empty.txt. Wenn Sie einen dieser Namen angeben, besteht kein Bedarf, auf dessen Vorhandensein zu testen. Diesen Teil des Skriptes habe ich während der Arbeit an diesem Artikel geändert. Vorher war es nur:

```
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 diesem Fall musste ich Dateien mit den Namen files_empty.txt und commands_empty.txt haben. Das war aber überhaupt kein Problem, da ich nur in einem Verzeichnis gearbeitet habe.
Am Ende kommt der Aufruf der Funktion 'doit'. Alles wird in dieser Funktion gesteuert. Diese Funktion enthält eine aus 'cat' und 'while' bestehende Schleife, die für jeden in der $hosts_file−Datei aufgeführten Rechner copy_files und execute_commands entsprechend der Modus−Variablen aufruft. So wird für jeden Rechner der Auftrag ausgeführt. 'host' enthält den aktuellen Rechnernamen oder die IP−Adresse.
Wir sollten uns nun die Funktion copy_files ansehen. Diese Funktion testet zuerst, ob der Wert von 'files_file' gleich 'files_empty.txt' ist oder nicht. Bei Gleichheit wird nichts getan. Wenn nicht, dann enthalten die Variablen 'direction', 'file1' und 'file2' für jede Zeile aus '$files_file' die Kopierrichtung und den Namen der ersten bzw. zweiten Datei. Entsprechend der Kopierrichtung wird der Kopiervorgang mittels ssh durchgeführt.
Zum Schluss sollten wir uns anschauen, was in der Funktion execute_commands passiert. Die Funktion testet, ob der Wert von 'commands_file' gleich 'commands_empty' ist oder nicht. Bei Gleichheit geschieht wieder nichts. Wenn nicht, dann wird jeder Befehl aus der '$commands_file'−Datei auf den entfernten Rechnern mittels ssh im Hintergrund ausgeführt. Nach der Ausführung des ssh−Befehls wird wait mit dem Parameter '$!' aufgerufen. Dieser Befehl stellt sicher, das jeder Befehl nach dem anderen ausgeführt wird. '$!' expandiert zur Prozess−ID des zuletzt ausgeführten Hintergrundbefehles.
Das wars. Einfach, nicht wahr?

Einfache Sicherung Ihrer Konfigurationsdateien

Hier ist eine erweiterte Anwendung des Skriptes. Sinn ist es, eine Sicherung der Konfigurationsdateien Ihrer Rechner oder Server vorzunehmen. Für diesen Zweck habe ich ein kleines Skript geschrieben, das aiinstall. sh benutzt:

```
#!/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
```

Sie müssen ein Verzeichnis namens servers einrichten. Unter diesem Verzeichnis muss es zwei Dateien geben: files_getbackup.txt und ll_servers.txt. Hier ist 'files_getbackup.txt':

r2l /root/backup.tgz backup.tgz

'll_servers.txt' enthält die Namen oder IP−Adressen der zu sichernden Rechner. Jeder in der Datei 'll_servers.txt' enthaltene Rechner muss ein Verzeichnis mit dem gleichen Namen haben und unter diesem Verzeichnis muss eine Datei namens commands_make_backups.txt vorhanden sein, die einen Befehl enthält, um aus den Konfigurationsdateien auf dem Rechner ein /root/backup.tgz−Archiv zu erstellen. Alle Sicherungen dieses Rechners werden unter diesem Verzeichnis gespeichert. Wenn ll_servers.txt folgenden Inhalt hat:

```
fileserver
dbserver
10.10.10.1
appserver
```

dann muss die Verzeichnis−Struktur Ihres 'servers'−Verzeichnisses wie folgt aussehen:

```
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
```

Und hier sind einige Beispiele für die Datei commands_make_backups.txt:

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

Die obige commands_make_backup.txt wird zur Sicherung von samba−, atalk− und Nameserver−Konfigurationen und Zonendateien benutzt.

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

Diese commands_make_backup.txt wird zum Sichern einer Apache−Serverkonfiguration und der Konfigurationsdateien genutzt.

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

Die obige commands_make_backup.txt wird zur Sicherung der Squid−Proxy−Serverkonfiguration und der Konfiguration von sekundären DNS−Servern benutzt.

Unter Benutzung des obigen Skriptes und dem Erstellen entsprechender commands_make_backup.txt−Dateien gemäß Ihren Bedürfnissen können Sie Sicherungen Ihrer Serverkonfigurationen vornehmen.

Schlussfolgerung

Das aiinstall.sh−Skript erlaubt Ihnen die Automatisierung einiger System−Administrations−Aufgaben. Das Skript basiert auf einer einfachen Benutzung der ssh−Programme. Sie werden dieses Skript schätzen, wenn es eine große Anzahl gleicher Rechner zu verwalten gibt.

Referenzen

- SSH, The Secure Shell: The Definitive Guide, von Daniel J. Barrett und Richard Silverman.
- Durch den Tunnel, von Georges Tarbouriech.
- Shell Programmierung, von Katja und Guido Socher.

Webpages maintained by the LinuxFocus Editor team © Erdal Mutlu "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Translation information: en −−> −− : Erdal Mutlu <erdal(at)linuxfocus.org> en −−> de: Hermann−Josef Beckers <beckerst/at/lst−oneline.de>

2005−01−11, generated by lfparser_pdf version 2.51