There are no translations available.
En esta tercera y última entrega veremos scripts sacados de un entorno de producción real que pueden ser usados por cualquier administrador de sistemas...

Tutorial Shell Scripts III

En esta última entrega veremos los siguientes puntos:

6. SCRIPTS EN LINUX: Un paso adelante : SSH sin contraseña, RSYNC y AWK.
7. SCRIPTS EN LINUX: Ejemplos de Scripts II.
   7.1 Script para automatizar la réplica de una Base de Datos MySQL
   7.2 Script para la creación de usuarios en varias máquinas remotas

6. SCRIPTS EN LINUX: Un paso adelante: SSH sin contraseña, RSYNC y AWK.

Si habeis seguido el monográfico paso a paso, ya tendreis algunas nociones de teoría y habreis visto algunos ejemplos de scripts. Si ya nos sentimos comodos y no estamos muy perdidos, entonces estamos listos para dar un paso adelante. En este artículo del monográfico vamos a tratar, un poco por encima, tres nuevos aspectos teoricos. Vamos a ver como se puede realizar una conexión ssh entre dos equipos sin tener que introducir la contraseña y vamos a ver en que consisten Rsync y AWK, dos herramientas de los entornos Unix que nos darán nuevas posibilidades.

SSH sin contraseña.

No os asusteis, esto no significa que vayamos a tirar la seguridad de nuestra red por el suelo. Muy al contrario. Lo que vamos a hacer es realizar una conexión entre dos equipos usando ssh, como método de autenticación usaremos un sistema clave publica / clave privada. Lo dicho... ¡empezamos!.

Debemos determinar **que usuario de que máquina** se conectara a **que usuario de que otra máquina** .

Debemos estar logados como "usuario a" en la "máquina a". Una vez en esta situación debemos hacer:

ssh-keygen -t rsa

Este comando nos creara una serie de archivos en la carpeta .ssh del home del "**usuario a".**
Uno de esos ficheros se llamará:
$HOME/.ssh/id_rsa.pub

Ahora debemos logarnos en el "equipo b" como "usuario b" y hacer un ftp al "equipo a".
Cuando se nos solicite el user/password para el ftp introducimos los del equipo a.

ftp >	**lcd /tmp**
ftp>	**mget /home/usuario_a/.ssh/ id_rsa.pub**
ftp>	**bye**

Ahora ya tenemos el fichero en la carpeta /tmp del "equipo b".

cat /tmp/id_rsa.pub >> $HOME/.ssh/authorized_keys

Con el comando anterior copiamos el contenido del fichero a un Nuevo fichero en la carpeta .ssh del usuario.

El último paso es dar los siguientes permisos:

chmod 0700 $HOME/.ssh/
chmod 0600 $HOME/.ssh/authorized_keys

Ahora que ya hemos terminado la tarea, debemos probar que todo ha ido bien.

Desde el **equipo_a** y como **usuario_a hacemos:**

ssh usuario_b@equipo_b

Esto debe permitirnos acceder sin contraseña. Debemos tener en cuenta que la primera vez que lo hagamos se nos solicitará incluir el equipo remoto en el fichero known_hosts. Tecleamos yes y tenemos concluida la tarea.

¿Que nos permite esto? Pues muy facil, nos permite ejecutar comandos en máquinas remotas, y nos permite hacerlo en un script programado en el tiempo, ya que no requiere que ningún administrador inserte la contraseña en la máquina remota .Un ejemplo, gracias a esta

forma de conexión podriamos centralizar la ejecución de copias de seguridad sin tener que programar todas ellas. Programariamos un script en un servidor que fuese lanzando las copias de seguridad en todos los demas y capturando las "salidas" de las ejecuciones en un solo log.

Veremos también como gracias a esta utilidad podemos adecuar el script que ya hemos visto de creación de usuarios para que sirva para crear usuarios en máquinas remotas.

RSYNC

Es una utilidad típica de los sistemas UNIX. Lo primero que debemos hacer es saber si la tenemos instalada ya en nuestra distribución linux. Lo más facil para saber si esta instalada es ejecutarla desde linea de comandos y ver que pasa. Si está instalado nos saldrá el típico mensaje de créditos y opciones de uso:

```
javi$ rsync
rsync version 2.6.3 protocol version 28
Copyright (C) 1996-2004 by Andrew Tridgell and others

Capabilities: 64-bit files, socketpairs, hard links, symlinks, batchfiles,
inplace, IPv6, 32-bit system inums, 64-bit internal inums


rsync comes with ABSOLUTELY NO WARRANTY. This is free software, and you
are welcome to redistribute it under certain conditions. See the GNU
General Public Licence for details.


rsync is a file transfer program capable of efficient remote update
via a fast differencing algorithm.


Usage: rsync [OPTION]... SRC [SRC]... [USER@]HOST:DEST
or rsync [OPTION]... [USER@]HOST:SRC DEST


....
```

.....

Pero bueno, al grano, ¿qué es Rsync? o mejor aun ¿para qué sirve?. Rsync nos va a permitir sincronizar carpetas dentro de una misma máquina o en máquinas diferentes. Esto, que así dicho no parece muy espectacular es muy útil.

Aunque la cantidad de usos y parametros para usarlo es enorme , la manera más frecuente de usarlo es:

rsync -avz user_remoto@ip_serv_remoto:/carpeta_remota /carpeta_local

Con el anterior comando lo que hacemos es sincronizar la carpeta del servidor remoto con nuestra carpeta local.

Estas dos operaciones las haremos usando el algoritmo que contiene rsync. Nos traeremos o llevaremos los ficheros y carpetas que no existan en la carpeta destino, o que aún existiendo sean versiones anteriores. Imaginaos lo útil que es esto para hacer copias de seguridad. Basta con que programemos un rsync para asegurarnos que una carpeta en un equipo tendrá siempre el mismo contenido que otra carpeta en otro equipo.

Con respecto al tema de las comunicaciones, rsync tiene una buenisima funcionalidad ya que permite que lo utilices sobre una conexión ssh (cifrada). Para hacerlo basta con usarlo de la siguiente manera:

rsync -avz -e 'ssh -i /user/.ssh/id_rsa' user@ip_serv_remoto:/carpeta_remota /carpeta_local/

ej: rsync -avz -e 'ssh -i /root/.ssh/id_rsa' root@192.168.17.100:/tmp/bbdd_dump /tmp/ >> /root/scripts/im

Con esta forma de usarlo, al montarlo sobre una conexión ssh, podremos efectuar la operación sin tener que introducir la contraseña del sistema remoto (siempre que hayamos hecho una distribucion de clave pública). Esto nos permitirá usar rsync en nuestros scripts programados. En los ejemplos siguientes hay un ejemplo muy claro del uso del rsync para, por ejemplo, automatizar copias de seguridad o respaldo de bases de datos.

AWK

AWK es en si mismo todo un lenguaje de programación diseñado para tratar con textos. No nos adentraremos mucho en él (tiene infinitas posibilidades). Veremos como podemos usarlo para tratar cadenas de texto en nuestros scripts.

Supongamos un caso en el que queramos obtener la fecha en el sistema de unos ficheros en un directorio. Todos sabemos que haciendo en la shell:

 # ls -lrta

obtenemos algo como:

```
root@ubuntu:~# ls -lrta
total 400
-rw-r--r-- 1 javi javi 580 Sep 5 2005 htaccess.txt
lrwxr-xr-x 1 root javi 54 Jan 11 2007 Enviar registro -> /Users/javi/Library/Assistants/Send Registration.s
drwxr-xr-x 5 javi javi 170 Jan 11 2007 Public
drwxr-xr-x 5 javi javi 170 Jan 12 2007 Sites
-rw-r--r-- 1 javi javi 3 Jan 12 2007 .CFUserTextEncoding
drwx------ 17 javi javi 578 Mar 16 2007 .aMule
-rw-r--r-- 1 javi javi 2611 Mar 22 2007 configuration.php
-rw-r--r-- 1 javi javi 14451 Mar 30 2007 template_css.css
drwxr-xr-x 3 javi javi 102 Apr 1 2007 PoisonDownloads
-rw-r--r-- 1 javi javi 9455 Apr 3 2007 upload.gallery.php
-rw-r--r-- 1 javi javi 3885 Apr 3 2007 show.gallery.html.php
-rw-r--r-- 1 javi javi 3332 Apr 3 2007 authEA.php
```

drwxr-xr-x 3 javi javi 102 Apr 20 2007 Shared

Si sobre la salida del comando invocamos awk, podemos hacer algo como:

root@ubuntu:~# **ls -lrta |awk '{print ($6)}'**

Sep
Jan
Jan
Jan
Jan
Mar
Mar
Mar
Apr
Apr
Apr
Apr
Apr

Es decir, le estamos diciendo que, sobre la salida del primer comando , imprima solo la 6ª columna. Podemos imprimir varias columnas haciendo:

root@ubuntu:~# **ls -lrta |awk '{print ($6 $7)}'**

Sep5
Jan11
Jan11
Jan12
Jan12
Mar16
Mar22

Mar30
Apr1

Tambien podemos agregar "literales":

```
root@ubuntu:~# ls -lrta |awk '{print ("mes="$6" dia=" $7)}'
mes= dia=
mes=Sep dia=5
mes=Jan dia=11
mes=Jan dia=11
mes=Jan dia=12
mes=Jan dia=12
mes=Mar dia=16
mes=Mar dia=22
mes=Mar dia=30
mes=Apr dia=1
```

Si queremos tratar un fichero con información tabulada, en el que sabemos que en lugar del "espacio" se ha usado un separador como "|" ó ":" ó "," podemos especificarlo. Supongamos el siguiente fichero:

```
root@ubuntu:~# cat fich_separadores
javier:martinez:7-8-1979:Sevilla
cristina:cacho:31-3-1979:Madrid
luis:lopez:2-2-1982:barcelona
```

Podemos generar una salida con el nombre y la ciudad de nacimiento sin más que indicarle al awk que el separador de campos es el caracter ":" :

```
root@ubuntu:~# cat fich_separadores |awk -F":" '{print ($1 " " $4)}'
javier Sevilla
```

cristina Madrid
luis barcelona

Veremos un ejemplo de un uso útil del AWK en uno de los scripts que vienen a continuación.

7. SCRIPTS EN LINUX: Ejemplos de Scripts II.

Ahora vemos dos scripts interesantes que incluyen la última tería vista. Considero que son muy útiles y que pueden haceros la vida más fácil si los adaptais a vuestras necesidades. ¡Vamos alla!.

7.1 Script para automatizar la réplica de una Base de Datos MySQL

Supongamos un escenario en el que tenemos dos servidores distintos. Uno de ellos tiene una aplicación con su correspondiente base de datos (puede ser una página web...). El otro servidor lo tenemos como servidor de emergencia. es decir, tenemos una copia de la aplicación y la Base de Datos, listo para ocupar el puesto del otro si se rompe o si hay que someterle a mantenimiento.

Si la aplicación sufre pocas modificaciones o pocos "despliegues" nuevos, no será dificil abordar la tarea de realizar las modificaciones en los dos servidores cada vez. Sin embargo la Base de datos tendra muchas inserciones y modificaciones cada día. Con el siguiente script podremos automatizar la tarea de llavar parejas las dos bases de datos.

Tenemos dos servidores. El servidor A es el de producción, y el servidor B es el que queremos mantener igual que el de producción. Antes de que este script pueda funcionar deberemos haber preparado la conexión del equipo A con el B a traves de SSH basado en clave publica / claver privada.

SERVIDOR A. **/root/scripts/export_bbdd.sh**

Almacenamos la fecha en una variable, luego la usaremos

```
FECHA=`date '+%d_%m_%y'`



### Nos situamos en el directorio donde guardaremos el export de la BBDD
cd /tmp/bbdd_dump

#### Borro los export que pudiese haber almacenados
rm -Rf export_*



#### Genero el export, componiendo el nombre del fichero con la fecha. Es decir el fichero de ex
mysqldump -uroot -pcontraseña --add-drop-table nombre_bbdd > /tmp/bbdd_dump/export_$FEC



####Controlo los posibles errores en la ejecución. Esto equivale a preguntarle al sistema si ha id
if [ $? -eq 0 ]
        then
                echo La BBDD se ha exportado con exito. >> /root/scripts/export.log
        else
                echo ERROR EN LA EXPORTACION DE LA BBDD!!! LA SALIDA DEL COMAND
                exit
fi



#####Escribo una marca con la fecha en el log del proceso.
echo "El export del dia $FECHA se ha realizado con exito" >> /tmp/bbdd_dump/export.log

#### Una vez hecho el export, puedo hacer que empiece la importación en la maquina remota.
ssh        root@192.168.1.102 '/root/scripts/import_bbdd.sh' >>/tmp/bbdd_dump/export.log
```

SERVIDOR B. **/root/scripts/import_bbdd.sh**

```
echo >> /root/scripts/import.log
echo "################################################################" >> /root/script
echo >> /root/scripts/import.log
```

```
FECHA=`date '+%d_%m_%y'`
echo FECHA DE LA EJECUCION: $FECHA >> /root/scripts/import.log



######Nos situamos en el directorio desde donde importaremos el fichero sql que se ha gener
cd /tmp/bbdd_dump
rm -rf *



###### Control de errores
if [ $? -eq 0 ]
        then
                echo El borrado del sql anterior se ha realizado con exito. >> /root/scripts/impo
        else
                echo ERROR EN EL BORRADO DE FICHEROS!!! LA SALIDA DEL COMANDO H
                exit

fi



##### Aquí nos traemos los ficheros que se han generado en el primer proceso.
rsync -avz -e 'ssh -i /root/.ssh/id_rsa' root@192.168.17.100:/tmp/bbdd_dump /tmp/ >> /root/scripts

##### Control de errores
if [ $? -eq 0 ]
        then
                echo El rsync se ha realizado con exito. >> /root/scripts/import.log
        else
                echo ERROR EN EL RSYNC!!! LA SALIDA DEL COMANDO HA SIDO DISTINTA
                exit

fi

######Importación en la bbdd del fichero que nos hemos traido.
mysql -uroot -pcontraseña --database nombre_bbdd < /tmp/bbdd_dump/export*.sql >> /root/scrip

if [ $? -eq 0 ]
        then
                echo La importacion del sql se ha realizado con exito. >> /root/scripts/import.l
        else
                echo ERROR EN LA IMPORTACION DEL SQL. LA SALIDA DEL COMANDO HA
                exit
```

```
fi
```

En general, si usamos un script para una tarea medianamente crítica es importante redirigir la salida del mismo a un fichero de log y controlar bien los errores generando comentarios en el log que nos permitan conocer o saber los distintos pasos "correctos" que ha ido dando nuestro script.

7.2 Script para la creación de usuarios en varias máquinas remotas

Este script es muy parecido al que ya habiamos visto. Para poder ejecutarlo automáticamente contra máquinas remotas deberemos haber preparado las conexiones ssh basadas en clave pública / clave privada desde el servidor que ejecuta el script a los servidores donde queremos administrar (crear) los usuarios.

```
echo ----------------------------------------------------------------------------
echo -------□□SCRIPT PARA LA ADMINISTRACION CENTRALIZADA DE USUARIOS□□----------
echo ----------------------------------------------------------------------------
echo

############################################################
###FUNCION QUE RECOGE LOS DATOS#############################
############################################################
#
#
peticion_datos()
{
######## "Limpio" los posibles valores de las variables que voy a usar.


unset $user
unset $grupo1
unset $grupo2
unset $shell
unset $password
unset $casa
```

######### Comienzo a pedirle datos al usuario. En muchos de los campos se le ofrecen opcione

```
echo Nombre de usuario:
read user
echo Identificador de Usuario
read ID
echo Grupo principal:[users]
read grupo1
if [ "$grupo1" = "" ]
then
        grupo1=users
fi
echo Grupo Secundario:
read grupo2
if [ "$grupo2" = "" ]
then
        grupo22=""
else
        grupo22="-g 'grupo2"'
fi

echo Shell:[/bin/bash]
read shell
if [ "$shell" = "" ]
then
        shell=/bin/bash
fi
echo Home del usuario: [/home/$user]
read casa
if [ "$casa" = "" ]
then
        casa=/home/$user
fi
echo Existe el home del usuario: [n]
read exist
if [ "$exist" = "" ] || [ "$exist" = "n" ]
then
        param=-m
fi
```

######### Aquí se le pide la password. Esto ya lo habiamos visto en un script anterior. Hace falt

```
echo Password:
read password
contrasena=`perl /root/scripts/crypt.pl $password`



########Se muestra la contraseña encriptada para comprobar que el script de perl está funciona



echo ____________________________
echo ----- DEBUG-------------
echo ____________________________

echo $contrasena



}

#######################################################################
###  FIN DE LA FUNCION  ##############################################
#######################################################################



#######################################################################
###  FUNCION QUE GENERA EL COMANDO  ########################
#######################################################################



generar_comando()
{
                comando="/usr/sbin/useradd -u $ID -G $grupo1 $grupo22 -d $casa $param -p

}

#######################################################################
###  FIN DE LA FUNCION  ##############################################
#######################################################################
```

```
echo En esta opcion podra crear un usuario en maquinas remotas. Puede crearlo en una maquin
echo Para poder ejecutarse correctamente, la maquina remota debe tener dado de alta el usuario

echo
echo Especifique si desea realizar la operacion contra una maquina "(1)", contra una zona "(2)":



####Esta parte esta muy bien. Se le ofrece al usuario la opcion de ejecutar la accion contra un or
###### o contra una lista de ordenadores. Las listas son ficheros almacenados en un directorio
######caso denominamos zonas.



read opcion
case $opcion in
1)
echo inserte el nombre o la ip del servidor:
read server
peticion_datos
generar_comando
ssh root@$server "$comando"
echo $comando

;;

2)

echo Las zonas definidas son:

### Accedo a  al directorio de zonas y me quedo con los nombres de los ficheros que en el se e
cd zonas
echo `ls -lrta |grep "-r" |awk '{ print $9 }'`
echo
echo Seleccione la zona sobre la que actuar:
read zona

                        peticion_datos
                        for server in `cat $zona | awk -F"|" '{print $1}'`
#### Esto se interpreta: "Para cada ip almacenada en la primera columna del fichero, usando de
                        do
                        echo Estoy con la maquina $server ### Para controlar en la pantal
                        generar_comando
                        ssh root@$server "$comando"
                        echo $comando
```

```
                                done
;;


*)
        echo La opcion elegida no es correcta
        echo .....Saliendo.....
        exit
;;
esac
```

Si se ha seguido con detenimiento el script os dareis cuenta que las llamadas "zonas"deben definirse en un fichero almacenado en una carpeta /zonas que "cuelga" de la ubicación del script. Los ficheros de listas de servidores que se almacenan en dicha carpeta deben seguir la siguiente estructura:

```
192.168.1.33|Ordenador bbdd mysql
192.168.1.43|Servidor web
192.168.1.75|Servidor de correo
```

Viendo la estructura del script no os será nada dificil el ir añadiendo funcionalidades tales como borrar usuarios, consultar los usuarios de un servidor etc...

Bibliografía

http://es.wikipedia.org/wiki/Int%C3%A9rprete_de_comandos

http://www.microsoft.com/technet/prodtechnol/windowsserver2003/es/library/ServerHelp/44500063-fdaf-4e4f-8dac-476c497a166f.mspx?mfr=true

http://support.microsoft.com/kb/313565/es

http://www.robvanderwoude.com/batexamples_0.html#0

http://www.wikilearning.com/monografia/fundamentos_de_bash/5846

http://www.samba.org/rsync/

http://es.wikipedia.org/wiki/Rsync

http://cm.bell-labs.com/cm/cs/awkbook/

http://es.wikipedia.org/wiki/AWK

http://es.wikipedia.org/wiki/Secure_Shell

http://www.openssh.com/es

Para volver al principio del tutorial, vaya al siguiente enlace: Tutorial Shell Scripts I .