LinuxFocus article number 278
http://linuxfocus.org

Automatiser l’administration système avec ssh et scp

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

L´auteur:

Erdal est l’un des éditeurs Turcs de LF. Il travaille actuellement comme administrateur système pour Linotype Library. En tant qu’inconditionnel de Linux depuis ses années universitaires, il aime travailler et développer dans cet environnement.

Résumé:

Si vous devez administrer un grand nombre de systèmes Unix/Linux, vous devrez certainement créer quelques scripts pour automatiser certains processus. Vous avez sans doute remarqué le côté répétitif des tâches quotidiennes réclamées par chaque système. Vous envisagez donc une certaine automatisation. C’est particulièrement applicable à la maintenance de nombreux systèmes Unix/Linux configurés de manière identique. Dans cet article, je présenterai un moyen de réaliser cela à l’aide des utilitaires ssh.

Traduit en Français par:
Georges Tarbouriech
<gt(at)linuxfocus.org>

Introduction

L’idée consiste à trouver le moyen de copier des fichiers depuis la station sur laquelle je travaille vers un ensemble de stations et de serveurs, et ensuite d’exécuter des commandes sur ces machines, comme installer des rpm ou modifier des paramètres. Parfois, nous devrons d’abord exécuter des commandes sur ces machines et ensuite récupérer certains fichiers pouvant résulter de ces commandes.

Afin de comprendre cet article vous devez avoir des connaissances de base sur la programmation shell. Pour plus ample information, lisez l’article de LinuxFocus Programmation du shell de Katja et Guido Socher. Vous devez aussi connaître les utilitaires ssh comme ssh-keygen, ssh-add, ssh, scp ou sftp. Une version libre du protocole SSH existe sous Linux OpenSSH et contient tous ces utilitaires. Des pages de manuel sont également disponibles.

Pourquoi utiliser ssh?

La réponse est une question : pourquoi pas ? On peut utiliser rsh-rcp ou telnet-ftp, mais ils ne sont guère adaptés aux environnements non sécurisés comme Internet ou même Intranet. Ssh offre un moyen de crypter les communications entre deux hôtes d'un réseau non sécurisé. Je ne parlerai pas des implications concernant la sécurité lors de l'utilisation de ces outils. Voir l'article de Georges Tarbouriech Par le tunnel.
En réalité, j'ai même utilisé dans le passé des scripts basés sur telnet/ftp.

Copier des fichiers et des répertoires avec scp

Afin de copier un fichier depuis un répertoire local vers une machine distante, la commande suivante peut être utilisée :

scp /chemin/vers/le/fichier/fichier1 utilisateur@hôte_distant:/répertoire_distant/nouveau_fichier

Dans cet exemple le fichier nommé fichier1 est copié du répertoire local vers l'hôte distant (l'hôte distant peut être l'adresse IP ou le nom de la machine) dans /répertoire_distant sous le nom nouveau_fichier. Vous devez vous identifier en tant que 'utilisateur'. Si l'authentification est acceptée et que l'utilisateur distant possède les droits nécessaires, le fichier sera copié. Il est possible d'omettre le nom du fichier de destination. Dans ce cas, le fichier est copié avec le même nom. C'est simplement pour indiquer la faculté de changer le nom pendant la copie.
Le contraire est bien évidemment possible : on peut copier un fichier distant dans un répertoire local.

scp utilisateur@hôte_distant:/répertoire_distant/fichier /chemin/vers/répertoire_local/nouveau_fichier

La commande scp possède aussi une option très pratique, '-r' permettant de copier des répertoires de manière récursive.

scp -r utilisateur@hôte_distant:/répertoire_distant.

La commande ci-dessus copie le répertoire 'répertoire_distant' avec tous ses sous-répertoires et fichiers, de l'hôte distant vers le répertoire courant sous le même nom.

Note: Le démon sshd doit bien sûr être actif sur l'hôte distant.

Se loger à distance avec ssh

A la place de rlogin ou telnet, il est préférable d'utiliser la méthode la plus sûre, autrement dit ssh :

ssh erdal@helvetica.fonts.de

Selon votre configuration il vous sera réclamé soit un mot de passe soit une "passphrase". Ici, nous nous

connectons à l'ordinateur distant helvetica.fonts.de sous le nom d'utilisateur erdal. La commande ssh possède de nombreuses options susceptibles de répondre à vos besoins. Voir la page de manuel de ssh.

Exécuter des commandes avec ssh

Ssh permet d'exécuter des commandes sur l'ordinateur distant :

ssh erdal@helvetica.fonts.de df -H

C'est pratiquement la syntaxe d'un rlogin. La seule différence se situe après le nom de l'hôte. La commande ('df -H' dans cet exemple) est destinée à être exécutée sur la machine distante. La sortie de la commande s'affiche sur votre terminal.

Se connecter à une machine distante sans mot de passe

Au lieu d'une authentification par mot de passe, on peut utiliser une paire de clés (publique/privée). Vous devez générer la votre. L'utilitaire ssh-keygen est prévu à cet effet :

ssh-keygen -b 1024 -t dsa

Le nom de la clé privée doit être fourni. Normalement, le nom de la clé publique est le même avec l'extension '.pub'. Ici, '-b 1024' est le nombre de bits de la clé à créer. Si vous ne le spécifiez pas, la valeur par défaut sera utilisée. '-t dsa' indique le type de clé. Les valeurs disponibles sont 'rsa1' pour la version 1 du protocole et 'rsa' ou 'dsa' pour la version 2. Il est recommandé d'utiliser la version 2 de SSH. Toutefois, si vous possédez d'anciens serveurs qui ne supportent que la version 1, vous devrez spécifier '-t rsa1' et créer une autre paire de clés. Vous pouvez forcer ssh à utiliser l'un des deux protocoles en précisant '-1' ou '-2'.

Afin de bénéficier de la clé, vous devrez installer la clé publique sur l'ordinateur distant. Le contenu du fichier de la clé publique doit être copié ou ajouté au fichier $HOME/.ssh/authorized_keys ou $HOME/.ssh/authorized_keys2. Attention, ne mélangez pas les clés pour des versions différentes du protocole. authorized_keys est utilisé pour la version 1 du protocole et authorized_keys2 pour la version 2. Si votre clé publique est correctement installée, lors de la prochaine connexion à cette machine votre "passphrase" vous sera réclamée, sinon vous devrez fournir le mot de passe de l'utilisateur distant. Vous pouvez restreindre les connexions vers vos systèmes en n'utilisant qu'une authentification par clé publique en éditant le fichier de configuration de sshd. Le fichier se nomme /etc/ssh/sshd_config et 'PasswordAuthentication' est le paramètre à modifier. Passez la valeur de ce paramètre à no (PasswordAuthentication no) et redémarrez sshd.

Jusque là, tout va bien. Nous disposons d'un moyen sécurisé de copier et d'exécuter des commandes sur des systèmes distants. Toutefois, pour automatiser certaines tâches, nous ne devrions pas avoir à taper des mots de passe ou des "passphrases", sinon, point de salut. Une solution pourrait consister à écrire le mot de passe ou la "passphrase" dans chaque script, mais ce n'est vraiment pas une bonne idée. Le meilleur moyen est d'utiliser ssh-agent puisqu'il est prévu pour gérer les clés privées servant à l'authentification par clés publiques. Démarrez un agent :

ssh-agent $BASH

et ajoutez vos clés privées par les commandes

ssh-add .ssh/id_dsa

ou

ssh-add .ssh/identity

id_dsa est le fichier de clé privée DSA et identity est le fichier de clés privées RSA1. Ce sont les noms donnés par défaut lors de la génération de clés par ssh-keygen. Evidemment, votre "passphrase" sera réclamée avant que ssh-add n'ajoute votre clé au ssh-agent. Vous pouvez lister les clés ajoutées avec la commande :

ssh-add -l

Maintenant, si vous vous connectez à un serveur qui possède votre clé dans le fichier correspondant, vous n'aurez rien à taper ! Le ssh-agent prend en charge le processus d'authentification.

Si vous utilisez ssh-agent comme décrit ci-dessus, vous ne pouvez vous en servir que depuis le terminal qui l'a démarré. Pour pouvoir en bénéficier à partir de n'importe quel terminal, il reste encore un peu de travail. J'ai écrit le script suivant pour démarrer 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
```

Ce script contrôle l'existence d'un fichier nommé agent_info dans le répertoire home de l'utilisateur où se trouvent habituellement les fichiers relatifs à ssh. Dans notre cas le répertoire se nomme '.ssh/'. Si le fichier existe, l'utilisateur en est averti et un bref message lui indique ce qui peut être fait. Si l'utilisateur n'a pas encore démarré le ssh-agent il doit effacer le fichier et relancer le script. Le script exécute le ssh-agent et capture les deux premières lignes du fichier agent_info. Cette information sera mise à profit ultérieurement par les utilitaires ssh. Ensuite, le mode du fichier est changé pour que seul son propriétaire possède les droits de lecture et d'écriture.

Lorsque l'agent est actif vous pouvez y ajouter vos clés. Mais auparavant vous devez localiser le fichier agent_info afin de permettre aux utilitaires ssh de savoir où se trouve votre agent :

source ~/.ssh/agent_info ou . ~/.ssh/agent_info

Et ajoutez vos clés avec ssh-add. Vous pouvez ajouter aussi les lignes suivantes dans votre fichier .bashrc de manière à ce que le fichier agent_info soit localisé à chaque ouverture d'un terminal :

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

ATTENTION: Vous devez sécuriser l'hôte à partir duquel vous exécutez ssh-agent et le script que je vais décrire. Sinon, si quelqu'un accède à votre compte, il aura accès à tous les serveurs utilisant les clés ssh. Tout a un prix !

Le script

Il est temps maintenant d'expliquer comment nous allons automatiser quelques unes des tâches d'un administrateur. L'idée consiste à exécuter un ensemble de commandes sur un groupe d'hôtes donné et de récupérer ou d'envoyer des fichiers de ou vers ces hôtes. C'est le travail quotidien d'un sysadmin. Voici le 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
```

Sauvons le script sous le nom ainstall.sh (automated installation) et essayons de l'exécuter sans paramètres. Nous obtiendrons le message suivant :

./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.
```

Comme le dit le message, si vous ne souhaitez pas exécuter de commande, donnez le nom commands_empty.txt à l'argument commands.txt et si vous ne voulez pas transférer de fichier, donnez le nom files_empty.txt à l'argument files_file. Parfois, vous devrez seulement exécuter des commandes et d'autres fois vous aurez seulement à transférer des fichiers.

Avant d'expliquer le script ligne à ligne, prenons un exemple d'utilisation : supposons que vous ayiez ajouté un serveur DNS secondaire à votre réseau et vouliez l'inscrire dans le fichier /etc/resolv.conf. Pour des raisons de simplicité, admettons que tous les hôtes possèdent le même fichier resolv.conf. La seule chose à faire est donc de copier le nouveau fichier resolv.conf sur tous les hôtes.
Il vous faut d'abord une liste de vos hôtes. Nous allons les inscrire dans un fichier hosts.txt. Ce fichier contient un nom d'hôte ou une adresse IP par ligne. Voici un exemple :

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

```
#### 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
```

Comme vous pouvez le voir dans l'exemple, il est possible d'inscrire des noms FQDN (Fully Qualified Domain Name) ou simplement la partie concernant l'hôte. Il vous faut ensuite un fichier dans lequel inscrire les fichiers à transférer. Il exite deux types de transfert :

- De l'hôte local aux hôtes listés dans le fichier hosts.txt. C'est notre cas.
- De chaque hôte listé dans le fichier hosts.txt vers l'hôte local. C'est par exemple le cas de notre script de sauvegarde abordé plus tard dans cet article.

Les fichiers à transférer sont listés dans un autre fichier. Sauvons ce fichier sous le nom files_file.txt. Ce fichier est au format suivant : chaque ligne contient l'information de copie d'un seul fichier. Il existe deux directions de copie : l2r (local to remote - local vers distant) et r2l (remote to local - distant vers local). l2r correspond au cas d'un fichier copié de l'hôte local vers un hôte distant. r2l correspond au cas d'un fichier copié d'un hôte distant vers l'hôte local. Après le sens de copie nous trouvons deux noms de fichiers. Les champs sont séparés par un espace ou une tabulation. Le premier fichier est copié vers le second fichier selon la direction choisie. Le nom de fichier pour l'hôte distant doit correspondre au chemin complet, sinon il sera copié dans le répertoire home de l'utilisateur root. Voici notre files_file.txt :

```
##############################################################################
# The structure of this file is :
# - The meaning of the fields 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 corresponding hosts.
#
#       Note: the order of using local and remote specifies the direction
#       of the copy process.
##############################################################################
l2r     resolv.conf     /etc/resolv.conf
```

Comme vous le voyez, j'ai déjà inclus la description de la structure du fichier. J'ajoute habituellement une description de tous les fichiers files_file.txt que j'utilise. C'est une solution simple mais efficace pour documenter. Dans notre exemple, nous voulons copier le fichier resolv.conf sur un hôte distant sous le nom /etc/resolv.conf. Dans un but didactique, après avoir copié le fichier sur ses destinataires j'ai ajouté des commandes permettant de modifier le propriétaire et le groupe et d'afficher le contenu du

fichier transféré. Les commandes à exécuter sont placées dans un fichier séparé. Appelons-le commands_file.txt. Le voici :

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

Le fichier des commandes contient les commandes qui seront exécutées sur chaque hôte listé dans le fichier hosts.txt. Ces commandes sont exécutées de façon séquentielle, c'est-à-dire dans l'ordre du fichier.

Voilà, nous avons maintenant les fichiers requis par ce simple exemple. La seule chose qui reste à préciser, c'est l'option 'mode', ce qui signifie : lequel des deux fichiers commands_file.txt ou files_file.txt doit être traité le premier. Il est possible de transférer les fichiers listés dans files_file.txt et d'exécuter ensuite les commandes sur l'hôte cible : c'est le mode = 1. Le contraire, exécuter les commandes et transférer ensuite les fichiers, correspond au mode = 2. Vous pouvez maintenant exécuter le script avec les arguments requis comme suit :

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

Un petit "truc" : j'ajoute toujours un préfixe files_ à ces fichiers texte et je donne ensuite un nom descriptif court, comme files_resolvconf.txt. En fait, la même méthode que celle employée pour hosts.txt et commands.txt

Il est temps maintenant de donner quelques explications sur le script proprement dit. Le programme commence par vérifier le nombre d'arguments et si celui-ci n'est pas égal à 4, un message de syntaxe est affiché. Si le nombre d'arguments est correct, ces derniers sont affectés aux variables correspondantes. Ensuite, si le fichier '~/.ssh/agent_info' existe, il est "sourcé". Ce fichier contient les informations concernant l'agent ssh actif. Si vous n'utilisez pas d'agent, vous devrez taper un mot de passe ou une "passphrase", ce qui signifie : pas d'automatisation :) L'existence de chaque fichier (hosts, files, commands) est ensuite vérifiée. Un contrôle particulier est effectué pour les fichiers files_empty.txt et commands_empty.txt. Si vous avez déjà défini de tels noms, il n'est pas nécessaire de vérifier l'existence de ces fichiers. J'ai modifié cette partie du script lors de l'écriture de l'article. Auparavant, il contenait seulement :

```
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
```

Dans ce cas, je devais avoir des fichiers nommés files_empty.txt et commands_empty.txt. Ce n'était pas un problème dans la mesure où je ne travaillais que sur un seul répertoire.
A la fin, nous trouvons l'appel à la fonction 'doit'. Tout est contrôlé par cette fonction. Elle possède une

boucle basée sur ’cat’ et ’while’, qui pour chaque hôte listé dans ’$hosts_file’ appelle copy_files et exécute les fonctions _commands selon le ’mode’. Ainsi, le travail est effectué pour chaque hôte. La variable ’host’ contient le nom d’hôte courant et son adresse IP.
Regardons la fonction copy_files. Cette fonction vérifie d’abord si la valeur de ’files_file’ est égale à celle de ’files_empty.txt’. Si oui, rien n’est fait. Dans le cas contraire, pour chaque ligne dans ’$files_file’, les variables ’direction’, ’file1’ et ’file2’ contiennent le sens de copie, le premier et le second noms de fichier. Selon la valeur de la variable ’direction’, la copie est exécutée par scp.
Enfin, regardons ce qui est fait dans la fonction execute_commands. La fonction vérifie si la valeur de ’commands_file’ est égale à celle de ’commands_empty.txt’. Si oui, rien n’est fait. Si non, chaque commande dans ’$commands_file’ est exécutée en tâche de fond sur l’hôte distant par ssh. Après l’exécution de la commande ssh, nous trouvons un appel à la commande wait avec le paramètre ’$!’. Cette dernière contrôle que les commandes sont bien exécutées l’une après l’autre. ’$!’ incrémente l’ID du processus de la dernière commande exécutée en tâche de fond.
Et c’est tout. Simple, non ?

Simple sauvegarde de vos fichiers de configuration

Voici un usage plus élaboré du script. L’idée est de faire une sauvegarde des fichiers de configuration de vos hôtes ou serveurs. Dans ce but, j’ai écrit un petit script qui utilise 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
```

Il vous faut un répertoire particulier nommé servers. Dans ce répertoire, vous devez avoir deux fichiers : files_getbackup.txt et ll_servers.txt. Voici le fichier ’files_getbackup.txt’ :

r2l /root/backup.tgz backup.tgz

’ll_servers.txt’ contient les noms ou les adresses IP des hôtes à sauvegarder. Chaque hôte listé dans le fichier ’ll_servers.txt’ doit posséder un répertoire avec le même nom contenant un fichier nommé

commands_make_backups.txt, qui lui-même contient une commande destinée à créer une archive /root/backup.tgz des fichiers de configuration de cet hôte. Il faut aussi un répertoire nommé backup. Toutes les sauvegardes de cet hôte seront stockées dans ce répertoire. Si le fichier ll_servers.txt contient :

```
fileserver
dbserver
10.10.10.1
appserver
```

alors la structure de votre répertoire '$servers' doit être la suivante :

```
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
```

Et voici quelques exemples de fichiers commands_make_backups.txt :

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

Ce fichier commands_make_backup.txt est utilisé pour sauvegarder les configurations de samba, atalk ainsi que celles du serveur de nom et des fichiers de zone.

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

Ce fichier commands_make_backup.txt est utilisé pour sauvegarder la configuration et les fichiers d'un serveur apache.

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

Ce fichier commands_make_backup.txt est utilisé pour sauvegarder les configurations du serveur proxy squid et du serveur DNS secondaire.

En utilisant le script ci-dessus avec des fichiers commands_make_backup.txt correspondant à vos besoins, vous pouvez sauvegarder les configurations de vos serveurs.

Conclusion

Le script ainstall.sh permet d'automatiser certaines tâches d'administration. Ce script repose sur une utilisation simple des utilitaires ssh. Ce script est très pratique dans le cas d'un grand nombre de systèmes identiques.

Références

- SSH, The Secure Shell: The Definitive Guide, de Daniel J. Barrett et Richard Silverman.
- Par le tunnel, de Georges Tarbouriech.
- Programmation du Shell, de Katja et Guido Socher.
