LinuxFocus article number 278
http://linuxfocus.org

Automating system administration with ssh and scp

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

About the author:

Erdal is one of the LF's Turkish editors. Currently he is working as System Administrator for Linotype Library. Being a big Linux fan since his University years, he likes to work and develope in this environment.

Abstract:

If you have large number of Linux/Unix systems to administer, then you'll certainly need some scripts to help you automate some of your processes. You should have noticed during everyday work that you do the same or similar things on every system. Maybe you have thought of having some way of automating these processes. This is especially true for maintaining large number of Linux/Unix systems which are identically configured. In this article I will introduce a way of doing this using ssh utilities.

Introduction

The idea is to find a way of copying some files from the workstation I am sitting in front of to a number of workstations or servers, and later execute some commands on those machines, such as running rpm installation or changing some system options. Sometimes we would first need to execute some commands on those machines and afterwards get some files, which can be a result of executed commands.

In order to follow this article you will need some basic shell programming understanding. For more information about shell programming look at LinuxFocus' Shell Programming article by Katja and Guido Socher. You will also need knowledge of ssh utilities, such as ssh-keygen, ssh-add, ssh, scp or sftp. There is a free implementation of the SSH protocol under Linux: OpenSSH, which contains all these tools. Man pages are also available.

Why use ssh?

The answer is a question: Why not? One can use rsh-rcp or telnet-ftp, they are not suitable in unsecure environments such as the Internet and maybe intranet. Ssh provides secure encrypted communications between two hosts over an insecure network. I am not going to discuss security implications when using those tools. Have a look at Through the tunnel article by Georges Tarbouriech.
Actually, I have used a in the past scripts which were based on telnet/ftp.

Copying files and directories using scp

In order to copy one file from a local directory to a remote computer the following command can be used:

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

In this example file with name file1 is copied from the local folder to the remote host (remote_host can be the IP or name of the remote machine.) under /remotedir with a new name newfile. You are prompted to authenticate as 'user'. If the authentication was successful and the remote user has proper permissions, then the file will copied. One can omit the destination file name. In this case the file is copied with the same name. Simply, this means one can rename files during copy.
The opposite is also possible: One can copy a remote file to a local folder:

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

There is also a very nice feature of the scp command. You can copy directories recursively by specifieng the '-r' option.

scp -r user@remote_host:/remotedir .

The above command copies the 'remotedir' directory and all its directories and files from the remote host to the current working directory with the same name.

Note: It is assumed that you have sshd daemon running on the remote host.

Remote login using ssh

Instead of rlogin or telnet one can use the more secure way, which is ssh:

ssh erdal@helvetica.fonts.de

Depending on your configuration, you will be prompted to enter either password or passphrase. Here, we are connecting to remote computer helvetica.fonts.de with remote user account erdal.The ssh command has a number of options which can be used depending on your needs. Have a look at the man page of ssh.

Executing commands using ssh

There is a possibility to execute commands on the remote computer using ssh :

ssh erdal@helvetica.fonts.de df -H

It is pretty much a remote login syntax. The only difference is after the hostname part. The command (in this example 'df -H') is given to be executed on the remote machine. The output from the command is displayed on your terminal.

Connecting to a remote computer without password

Instead of using password authentication, one can use a key pair (public/private). You need to generate your key pairs. There is ssh-keygen utility, which can be used to generate keys for ssh:

ssh-keygen -b 1024 -t dsa

You will be asked for the name of the private key. Normally, the name of the public key is the same as the private key with '.pub' appended. Here '-b 1024' is the number of bits in the key to create. If you don't specify the default value will be used. The '-t dsa' is used to specify the type of the key. The possible values are 'rsa1' for protocol version 1 and 'rsa' or 'dsa' for protocol version 2. I would recommend using protocol version 2 of SSH. But if you have old servers which support only protocol version 1 you will need to specify '-t rsa1' and create another key pair. You can force ssh to use protocol version 1 or protocol version 2 by specifying '-1' or '-2' respectively.

In order to use your key, you should install your public key on the remote computer. The public key file contents should be copied or appended to the $HOME/.ssh/authorized_keys or $HOME/.ssh/authorized_keys2 file. Be careful and do not mix keys for different protocol versions. authorized_keys is used for protocol version 1. authorized_keys2 is used for protocol version 2. If you properly installed your public key, the very next time when you connect to that computer you will be first asked to enter your passphrase and if you fail, then you'll be prompted for the passowrd of the remote user. You can restrict connection to your systems to using only public key authentication by editing the configuration file of the sshd. The file name is /etc/ssh/sshd_config and the parameter that you have to change is 'PasswordAuthentication'. Change this parameter value to no (PasswordAuthentication no) and restart your sshd.

Untill this point everything is okay. We have a secure way of copying and executing commands on remote systems. But, for automating some jobs we should not have to type passwords or passphrases. Otherwise we cannot automate anything. A solution could be to write in every script required password or passphrase, which is not a good idea anyway. The better way is to use the key-agent for taking care of our passphrases. ssh-agent is a program to hold private keys used for public key authentication. You should start a key-agent :

ssh-agent $BASH

and add your private keys to it using the

ssh-add .ssh/id_dsa

or

ssh-add .ssh/identity

id_dsa is DSA private key file and identity is RSA1 private key files. These are the default file names given during key generation via ssh-keygen. Of course you will be asked for your passphrase before the ssh-add adds your key to the ssh-agent. You can list which keys are added by issue the following command:

ssh-add -l

Now, if you connect to a server which has your key in the authorized file, then you will be connected without typing anything! The ssh-agent will take care about the authentication process.

When you use ssh-agent as described above, you can only use it within the terminal that ssh-agent was started. If you want to use the ssh-agent from any terminal that you open, then you will have to do a little bit more work. I wrote the following small script for starting the 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
```

The above script checks for an existence of a file called agent_info under the users home directory where normally ssh user related files are. In our case the directory is '.ssh/'. If the file exists the user is warned of the existence of the file and is given a small message about what can be done. If the user is not already running the ssh-agent then he or she must delete the file and start the script again. The script runs the ssh-agent and captures the first two lines into the agent_info file. This information will be used from ssh utilities afterwards. Next there is a line to change the mode of the file, so that only the owner of the file can read and write to it.

When your agent is up and running you can add your keys to it. But before this, you must source the agent_info file, so that ssh tools know where your agent is:

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

And add your keys with ssh-add. You can add the following lines to your .bashrc file so that everytime you open a new terminal, you will automatically have the agent_info file sourced:

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

WARNING: You must secure your host from which you are using ssh-agent and the automated script which I am going to describe here. Otherwise, if anyone has access to your account, then he or she will have access to all servers that you have access with ssh keys. Everything has its price!

The script

Now it is time to explain how we are going to automate some jobs of a system administrator. The idea is to execute a set of commands for a given list of hosts and to get or put some files from or to these hosts. This is what system admins need to do frequently. Here is the 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
```

Let us save the script as ainstall.sh (automated installation) and try to run it without any parameters. We will get the following message:

./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.
```

As the message says if you do not want to execute any commands, then give commands_empty.txt name for commands.txt argument and if you do not want to transfer any files, then give files_empty.txt name for files_file argument. Sometimes you will need only to execute some commands, while other times only to transfer some files.

Before explaining the script line by line let me give an example usage: Suppose that you have added a secondary DNS server to your network and would like to add this to /etc/resolv.conf file. For simplicity suppose also that all your hosts have the same resolv.conf file. So the only thing that you have to do is to copy the new resolv.conf file to all hosts.
First you need a list of your hosts. We are going to write all hosts in a file called hosts.txt. The format of the hosts.txt file is that every line contains only one host name or host IP. Here is an example:

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

As you see from the example, one can specify fully qualified host names or just the host part. Then you need a file where you will write files to be transfered. There is two types of transfer possible:

- From local host to hosts listed in the hosts.txt file. This is our case of transfer.
- From every host listed in the hosts.txt file to the localhost. This is when we will get some files from every host. For example the usage with our simple backup script, that I will describe later on in this article.

Files to be transfered are listed in another file. Let save this file as files_file.txt. The files_file.txt file's format is as follows: Every line includes information to copy only one file. There are two possible directions of copy : l2r (local to remote) and r2l (remote to local). l2r is when a file is copied from the localhost to a remote host. r2l is when a file from a remote host is copied to the localhost. After the direction keyword comes two file names. Fields are separated by a space or tab. The first file is copied to the second file according to the direction keyword. The file name for the remote host should be fully qualified, otherwise it will be copied to the home directory of the user root. Here then is our 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
```

As you see I have already inlcuded a description of how the file is structered. Normally I include this description for every files_file.txt file that I use. It is a simple but good solution for documentation. In our example we want to copy resolv.conf file on a remote host as /etc/resolv.conf. For demonstration purposes after copying it to destination hosts I added commands to change the owner and group owner and display the contents of the /etc/resolv.conf. Commands to be executed are placed in a separate file. Let us call the commands files commands_file.txt. Here is our commands_file.txt:

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

The commands file contains commands that are going to executed on every host listed in the hosts.txt file. Commands are executed in a sequential manner, this means first is the first command executed and after the second, and so on.

Okay, now you have all needed files for this simple example. The only thing that is left to be specified is the ’mode’ option, which means which of the two files: commands_file.txt or files_file.txt must be processed first. One can transfer files listed in the files_file.txt file and then execute all commands on the target host, this is mode=1. And the opposite, execute commands and then transfer files, which is mode=2. Now you can execute the script with the needed arguments as follows:

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

A small tip: normally I always prefix files.txt with files_ and after that give a short descriptive name, like files_resolvconf.txt. The same technique I apply for hosts.txt and commands.txt.

Now it is time to explain a little bit about the script itself. The program starts checking the number of arguments and if it is not 4 then the usage message is displayed. If the number of arguments is right, then arguments are assigned to corresponding variables. Then if ’~/.ssh/agent_info’ file exists, it is sourced. This file contains information about your running ssh agent. If you do not use an agent, then you’ll have to enter passwords or passphrases manually, which means no automation:). Afterwards every file (hosts, files and commands) are tested for existence. There is also a special test for files_empty.txt and commands_empty.txt. If you did specify such a name, then there is no need to test for file existence. I have changed this part of the script during the writing of this article. Before, it was only :

```
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 this case I had to have files with name : files_empty.txt and commands_empty.txt. But it was not a problem at all, because I was working only in one directory.
At the end comes the call to the function ’doit’. Everything is controlled in this function. The function has a loop made with ’cat’ and ’while’, which for every host listed in the ’$hosts_file’ calls copy_files and execute_commands functions according to ’mode’. So for every host the job is done. ’host’ variable contains the current host name or IP address.
Let us look at the copy_files function. This function first checks if the value of ’files_file’ is equal to ’files_empty.txt’ or not. If it is equal nothing is done here. If not then, for every line in the ’$files_file’, ’direction’, ’file1’ and ’file2’ variables contain the direction of copy, first file name and the second file name respectively. According to the value of the ’direction’ variable, a copy is performed using scp.
At the end let us have a look at what is done in the execute_commands function. The function checks if the value of ’commands_file’ is equal to ’commands_empty.txt’ or not. If it is equal nothing is done here. If not then every command in the ’$commands_file’ is executed on the remote host using ssh in the background. After executing ssh command there is call to wait command with parameter ’$!’. This

command ensures that every command is executed one after the other. ’$!’ expands to the process ID of the most recently executed background command.
That’s it. Simple isn’t it?

Simple backup of your config files

Here is a more advanced usage of the script. The idea is to make a backup of configuration files of your hosts or servers. For this purpuse I wrote a small script that uses 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
```

You have to have a special directory called servers. Under this directory there must be two files: files_getbackup.txt and ll_servers.txt. Here is the ’files_getbackup.txt’ :

r2l /root/backup.tgz backup.tgz

’ll_servers.txt’ contains the names or IP addresses of the hosts to be backed up. Every hostname listed in the ’ll_servers.txt’ file must have a directory with the same name and under this directory must be a file named commands_make_backups.txt, which contains a command to make a /root/backup.tgz archive from the configuration files on that host. And a directory named backup. All backups of this host will be stored under this directory. If the contents of the ll_servers.txt is :

```
fileserver
dbserver
10.10.10.1
appserver
```

then the directory structure of your ’$servers’ directory must be as follows:

```
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
```

And here are some examples for commands_make_backups.txt files:

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

The above commands_make_backup.txt is used to backup samba, atalk and nameserver configurations and zone files.

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

The above commands_make_backup.txt is used to backup an apache server configurations and files.

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

The above commands_make_backup.txt is used to backup squid proxy server and secondary dns server configurations.

By using the above script and constructing commands_make_backup.txt files according to your needs you can make backups of your server's configurations.

Conclusion

The ainstall.sh script allows you to automate some system administration jobs. The script is based on a simple usage of ssh tools. You will appreciate this script when there are big number of identical systems.

Referencess

- SSH, The Secure Shell: The Definitive Guide, by Daniel J. Barrett and Richard Silverman.
- Through the tunnel, by Georges Tarbouriech.
- Shell Programming, by Katja and Guido Socher.

Translation information:
en --> -- : Erdal Mutlu <erdal(at)linuxfocus.org>

2005-01-14, generated by lfparser_pdf version 2.51