Anna Maria Lopez Lopez

Building scripts for automated FTP

AUTOMATIC DELIVERY

If you find your self executing the same few steps in FTP, you'll save time and effort with a little automation. **BY DAVID TANSLEY**

File Transfer Protocol (FTP) is the defacto standard for transferring files over TCP/IP networks. The basic FTP protocol has been around for years, and though FTP is not especially secure, it will be around for many years to come due to its many features and well established user base. This article examines some techniques for automating FTP through scripting. I will not be focusing on the different types of FTP servers like wu-ftp or vs-ftp but will concentrate instead on the client side. I'll show you how you can use a script to connect to an FTP server and retrieve files.

Part of this discussion will include a look at simple login security, but keep in mind that the level of security you'll need for your own network will depend on the policies of your organization. It is possible, for instance, to use FTP with a secure VPN-style tunnel. You can find secure versions of FTP that offer built-in encryption for communication as well as encryption. I will leave such details for your own network and will focus, instead, on the business of automating file transfer.

Using .netrc

Whether you are operating FTP in batch mode or interactive mode, it is often better to have a mechanism that lets you login automatically to a remote site. Before we start into the details of FTP automation, I'll begin with a look at automated login. A special file called *.netrc*, which is located in your *$HOME* directory, allows you to automate login in FTP. The file permissions for *.netrc* should be set to read/write for the owner only – *chmod 600*. The file can contain multiple entries for remote sites. When a *.netrc* file is present, if the machine name given in the FTP command line matches the name of a machine in *.netrc*, FTP will use the login and password associated with the *.netrc* entry.

The format of the *.netrc*file comes in two flavors. I prefer the following format:

```
machine <remote_host> login <login> password <password> password <account_password>
```

Each remote host receives a single entry in the file. The password is the password

Listing 1: .netrc file

```
machine ftp.emea.ibm.com login anonymous password david.tansley@btinternet.com
machine uk01lx6001 login   dxtans      password l0opy
machine uk04lx6003 login   dxtans password        mas123
```

Listing 2: ftp1

```
#!/bin/bash
# ftp1
ftp -i -v <<mayday
open uk01lx6001
ascii
lcd /tmp
cd /etc
get hosts
quit
mayday
```

you need to connect to the remote host. The account password is only useful if the remote host requires another authentication process. Typically, only the first three entries are necessary.

Listing 1 is a *.netrc* file with three entries. The first entry is a web FTP public server. The login of *anonymous* generally means that files can be uploaded or downloaded from a publicly accessible folder. Though it is considered good form to accompany *anonymous* login with your email address as the password, this rule is not enforced. The other entries in Listing 1 are hosts on an internal network.

To connect to the host uk01lx6001, you just need to type in the following:

```
$ ftp uk01lx6001
```

As discussed earlier, when FTP starts, it will look for the *.netrc* file. In this case, since I am connecting to the host uk01lx6001, the client will search *.netrc* for the host name uk01lx6001; it will then use the login and password to connect to the remote server.

Getting a File

We can now put together a rather simple FTP script that connects to a remote host and grabs the */etc/hosts* file (see Listing 2). Note that FTP is invoked with the interactive mode turned off and verbose mode turned on. Using the *here document* approach denoted by the < < symbol, we specify that everything between the first and second occurrence of the word *mayday* will be taken as read from standard input. First we open a connection to the host uk01lx6001; we inform FTP that we will be using ascii for the transfer, then we change the directory on the local side to */tmp*, and change the remote directory to */etc*. Next, we get the *hosts* file across, so that it will reside in */tmp*. We then quit *FTP*. The second occurrence of the word *mayday* terminates our standard input, and thus, as there are no more lines of code, the script will exit.

Running the code in Listing 2 gives the output shown in Listing 3. Notice that, by default, FTP will try to perform the transfer in binary. You can change this behavior by issuing the *ascii* command. Also notice in the output that we are indeed initiating FTP in passive mode.

Files in a List

You can use a here document to parse a list of files to transfer, however, using this method means you have to connect and terminate for each transfer. Listing 4 demonstrates this technique. You may be thinking you could put a *for* loop inside the FTP code block, but you cannot, as you will be actually connected to the remote host in FTP, not in the bash shell.

Common FTP Options

Before delving deeper into how to automate FTP, let's look the most common FTP options:

-p: specifies that the connection should be in passive mode. You are the client, so definitely use passive mode for data transfers. Passive mode is the preferred approach when behind a firewall because the client tells the server which port to use. This is the default now, but some older versions still do not use passive as the default.

-i: turns off interactive prompting when negotiating FTP transfers. If you know what files you need to get or put, this option is invaluable.

-n: stops FTP from attempting auto login with the initial connection. If auto login is enable, FTP checks the *.netrc* for login information. If no match for the host is found in *.netrc*, FTP will then prompt for the login/password.

-v: sepcifies verbose mode. This option tells FTP to show what is happening by being very verbose.

.netrc Tricks

In some cases, you may wish to check for the presence of the .netrc file before

Listing 3: Output of ftp1

```
$ ftp1
Connected to uk01lx6001 (168.14.2.4).
220 uk01lx6001 FTP server (Version 4.1 Wed Mar 26 16:45:44 CST 2003) ready.
331 Password required for dxtans.
230-Last unsuccessful login: Tue Jan 18 12:18:34 GMT 2005 on /dev/pts/0
230-Last login: Tue Mar 29 18:40:33 BST 2005 on ftp from ::ffff:168.14.2.9
230 User dxtans logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
200 Type set to A; form set to N.
Local directory now /tmp
250 CWD command successful.
local: hosts remote: hosts
227 Entering Passive Mode (162,14,2,4,209,161)
150 Opening data connection for hosts (2370 bytes).
226 Transfer complete.
2443 bytes received in 0.00135 secs (1.8e+03 Kbytes/sec)
221 Goodbye.
```

Listing 4: ftp2

```
#!/bin/bash
# ftp2
list="hosts hosts.allow hosts.deny"
for files in $list
 do
   ftp -i -v <<mayday
   open uk01lx6001
   ascii
   lcd /tmp
    cd /etc
    get $files
    quit
    mayday
done
```

initiating an FTP session. For instance, you may wish for the script to follow one authentication path if *.netrc* is present and perform a different form of authentication if *.netrc* is missing. Keep in mind that it is a good idea to use *.netrc* whenever you can. Do not rely on interactive authentication if you don't have to, and for heaven's sake, do not hard code the password into an FTP script. Listing 5 contains a simple test block that checks to see if the *.netrc* file is present and readable by the script; if the test reveals that the *.netrc* file is present, the script presents the file to standard output.

If you want to offer the option of allowing the user to use one of the *.netrc* entries, it is sometimes convenient to present the contents of the file to the user, so that the user can select which record entry to use. One way of presenting the *.netrc* file to the user is to *cat* the file to standard output using the *-n* option, which will number each line. Listing 6 presents a framework showing how a solution using the *-n* option could be achieved. First we check that the *.netrc* is readable and thus present. We can then determine how many records are in the *.netrc* file using the following command:

```
max_recs=`cat $netrc_file | awk 'END{print NR}'`
```

You may be wondering why we did not use the *wc -l* command, as this alternative approach would require less processing. Unfortunately, if we use the *wc -l* command, the command pumps the variable substitution full of spaces, which does not look good cosmetically when the file is displayed to standard output.

Listing 5: ftp4

```
#!/bin/bash
# ftp4
netrc_file=$HOME/.netrc
if [ -r "$netrc_file" ]
then
 cat $netrc_file | awk '{print $2,$4}'
else
 echo "$netrc_file not present"
fi
```

If you use the *cat* command, all record entries are displayed to standard output, but only the host and login fields of each record appear. Listing 7 shows the output of this code.

We next prompt the user to enter a number that represents the record they wish to use, using the *$max_recs* variable, which holds the total number of records, as an upper limit. Once selected, the input number is used to extract the required fields from the record using awk's NR function. Following this, the results are echoed to standard output.

Listing 6 demonstrates one way to use a menu driven interface. I have not put in any serious error checking code, apart from a numerical range check, as I want to demonstrate that it does not take too much code to put together a menu-driven framework.

Checking for Errors

You will want to check for errors at the end of the FTP script. FTP provides quite a few error codes. Table 1 shows a few common return codes that may be considered errors or warnings.

One way to check for errors is to *egrep* at the end of a script run, as shown in Listing 9. The first thing we need to do is redirect all FTP output into a log file, as follows:

```
ftp -i -v  >> $log 2>&1 <<mayday
```

FTP will redirect all output including errors from the FTP session into a log file whose value is held in the variable *$log*. Once the FTP session has finished, we simply use egrep to include a list on codes or words we wish to match, with each pattern separated by the bar | sign. When you are using any grep command within a script, it is always a good idea to redirect the output of the pattern match to */dev/null*; this keeps unwanted messages from cluttering the standard output:

```
if egrep "202|421|426" $log > /dev/null 2>&1
```

Table 1: FTP Return Codes

202	Command not implemented
421	Service not available
426	Transfer aborted
450	File unavailable
500	Syntax error
501	Syntax error in arguments
503	User not logged in
550	File unavailable
553	Illegal filename
666	File or directory does not exist
777	unknown host
999	Invalid command

Listing 6: ftp5

```
#!/bin/bash
# ftp5
netrc_file=$HOME/.netrc
if [ -r "$netrc_file" ]
then
max_recs=`cat $netrc_file | awk 'END{print NR}'`
cat -n $netrc_file | awk '{print $1,": connect to [",$3,"] as user[",$5,"]"}'

 echo -n " Select record to use [ 1 .. $max_recs ] :"
  read ans
if [$ans -ge 1 ] && [ $ans -le $max_recs ]
then
   host=`cat $netrc_file | awk "NR==$ans"|awk '{print $2}'`
   user=`cat $netrc_file | awk "NR==$ans"|awk '{print $4}'`
   password=`cat $netrc_file | awk "NR==$ans"|awk '{print $6}'`
 else
   echo "invalid choice, needs to be [ 1 .. $max_recs ]"
   exit 1
fi     #  $ans in numeric range
 echo -e "selected info is:\nhost [$host]\nuser [$user]\npassword [$password]"
else
 echo " Sorry cannot read $netrc_file"
fi     # netrc present
```

If egrep returns *true* with any match, we simply exit with a 1 and echo a message to standard output and the log file; If egrep does not return a *true*, we exit with a 0 status. If you will be running scripts in batch mode, rather than interactively from the command line, you should not program the script to echo anything out to the standard output. If you are running batch mode scripts, you should instead write directly to a log file. If this is the case, you could amend the exit code to:

```
echo "Errors" >> $log
```

And likewise, for a good exit message, use the following:

```
echo "OK" >>$log
```

Conclusion

If you find yourself repeating the same few FTP commands over and over again for a recurring file transfer task, you can write a script to automate the file transfer.

Automated FTP is a very simple and productive means for transferring files between hosts. Do not be afraid of creating different FTP scripts to fill different administration requirements. The scripts are easy to write and easy to adapt. Whatever the purpose of the script, I do strongly suggest you use the *.netrc* file rather than embedding credentials in the script. ■

Listing 7: ftp7

```
#!/bin/bash
# ftp7
log=ftp.log
>$log

list="hosts telnet.conf"
host="uk01lx6001"

echo "Script name [ `basename $0` ]" >>$log
for files in $list
do
 ftp -i -v  >> $log 2>&1 <<mayday
 open $host
 ascii
 lcd /tmp
 cd /etc
 get $files
 quit
 mayday
done
 if egrep "202|421|426|450|500|501|503|550|553|666|777|999" \
$log > /dev/null 2>&1
  then
   echo "Errors" | tee -a $log
   exit 1
 else
  echo "OK" | tee -a $log
  exit 0
fi
```

advertisement