Linux Lessons

"**Linux Alternatives to FTP"**

by Pete Choppin

One of the many advantages of Linux/Unix is how many ways you can do one thing. This week's Linux Lesson is going to show you some of the alternatives to transferring files over a network connection.

In this column we will cover rsync, scp and tar. Please note that there are many other ways to use these commands. These are just some of the more common ones. The methods covered assume that ssh is used in all sessions. These methods are all much more secure and reliable than using rcp or ftp. This is also a great alternative for those looking for an FTP alternative to transferring files over a network.

scp

scp, or secure copy, is probably the easiest of all these methods. It is designed as a replacement for rcp, which was a quick copy of cp with network functionality.

Here is the scp syntax:

scp -Cpr username@example.net:destination_path

In the command, the -C switch compresses the data before it goes over the network, which significantly decrease the time it takes to copy large files. -p stands for "preserve," in other words, it keeps the last modified time, access time, etc. the same, and -r recursively copies subdirectories. Note: For the –r switch to work, you must specify the source file as a directory.

Before scp does any copying it first connects via ssh. Unless proper keys are in place, then you will be asked for usernames, so you will need the username and password to the destination connection. Otherwise, you can run this command on the source computer:

ssh-keygen -t dsa

This will create a pair of keys at $home_dir/.ssh/ and by default, the public key will be called id_dsa.pub.

What scp shouldn't be used for:

1. When you are copying more than a few files, as scp spawns a new process for each file and can be quite slow and resource intensive when copying a large number of files.

2. When using the -r switch, scp does not know about symbolic links and will blindly follow them, even if it has already made a copy of the file. That can lead to scp copying an infinite amount of data and can easily fill up your hard disk, so be careful.

rsync

rsync has very similar syntax to scp :

rsync -ave ssh source.server:/path/to/source /destination/dir

rsync's specialty lies in its ability to analyze files and copy only the changes made to files rather than all files. This can lead to enormous improvements when copying a directory tree a second time.

Switches:

-a Archive mode, most likely you should always keep this on; it preserves file permissions and does not follow symlinks.

-v Verbose, lists files being copied

-z Enable compression, this will compress each file as it gets sent over the pipe. This can greatly decrease time depending on what sort of files you are copying.

-e ssh Uses ssh as the transport; this should always be specified.

Disadvantages of using rsync :

1. Picky syntax; use of trailing slashes can be confusing.

2. Have to remember that you are using ssh.

3. rsync is not installed on all computers.

tar

tar is usually used for archiving files, but what we are going to do in this case is tar it, then pipe it over an ssh connection. tar handles large file trees quite well and preserves all file permissions, etc., including those Unix systems that use ACLs, and works quite well with symlinks.

The syntax is slightly different as we are piping it to ssh :

tar -cf – /some/file | ssh host.name tar -xf – -C /destination

or with compression:

tar -czf – /some/file | ssh host.name tar -xzf – -C /destination

Switch -c for tar creates an archive and -f which tells tar to send the new archive to stdout.

The second tar command uses the -C switch, which changes directory on the target host. It takes the input from stdin. The -x switch extracts the archive.

The second way of doing the transfer over a network is with the -z option, which compresses the stream, decreasing the time it will take to transfer over the network.

If using the -v (verbose) switch, be sure to include it only on the second tar command, otherwise you will see double output.

Using tar and piping can also be a great way to transfer files locally to be sure that file permissions are kept correctly:

tar cf – /some/file | (cd /some/file; tar xf -)

This may seem like a long command, but it is great for making sure all file permissions are kept intact. What it is doing is streaming the files in a sub-shell and then untarring them in the target directory. Please note that the -z command should not be used for local files, and no performance increase will be visible as overhead processing (CPU) will be evident, and will slow down the copy.

Downsides of using tar:

1. The syntax can be hard to remember

2. It's not as quick as to type scp for a small number of files

3. rsync will beat it hands down for a tree of files that already exist in the destination.

There are several other ways of copying over a network, such as FTP, NAS and NFS, but these all require specialized software installed on either the receiving or sending end, and hence are not as useful as the above commands.