SECURE RSYNC REPLICATIONS

There are many times in an administrators life when they want or -in most cases- need to do some sort of replication. Now you can use something like *ftp*, but using cleartext protocols is a bad thing. You could also use something like *sftp* or *scp*, this will be secure but the problem here is that each time you do the replication you will transfer the entire set of data. This is where *rsync* comes into play, this is a very handy tool that does bit-level replication. But isn't *rsync* a cleartext protocol you say? Yes it is, but it does allow for using *ssh* for it's replication process - natively. Very cool! Lets take a practical look at the entire process by setting up a replication process between two hosts.

What is needed?
Well quite simply, *rsync* and *ssh* must be available on each host. Then you will need to have a folder you want to replicate on the one host, and a folder on the second host that you want to replicate to. If you want to batch script the replication process you can setup the ssh login process to be automatic (see here or here). In our example, we have host *192.168.10.80* with the folder */admin/backup*, replicating to the */admin/backup* folder on *192.168.10.13*. Lets take a look at each system..

192.168.10.80	ls -l /admin/backup total 9448 -rw-r--r-- 1 root root 19 Jan 7 22:46 list.txt -rw-r--r-- 1 root root 2037760 Jan 7 22:43 nc.tar -rwxr-xr-x 1 root root 7611079 Jan 7 22:44 webmin-1.110-1.noarch.rpm
192.168.10.13	ls -l /admin/backup total 0

The first time
Now for any replication process, the first time it is run will generally be the longest. As this is the time that it copys over everything as nothing exists yet on the target system. Lets take a look at the command itself..

```
rsync -arpogtvv --stats --delete -e ssh /admin/backup/ 192.168.10.13:/admin/backup/
```

The command line options I have chosen are fairly general for replication...

a	this sets some archive functions
r	this means recursive
p	this keeps the permissions
o	this keeps the owner
g	this keeps the group
t	this keeps the timestamps
v	this means verbose (twice for more)
--stats	this produces some statistics
--delete	this will delete anything on the destination not on the source
-e	this specifies the shell the process should use
<source>	this is the folder the the process should sync from
<destination>	this is th folder (including host) that the process should sync to

Lets see what happens when I run it..

```
opening connection using ssh 192.168.10.13 rsync --server -vvlogDtpr --delete . /admin/
backup/
root@192.168.10.13's password:
building file list ...
expand file_list to 4000 bytes, did move
done
deleting in .
delta-transmission enabled
./
list.txt
```

```
nc.tar
webmin-1.110-1.noarch.rpm
total: matches=0  tag_hits=0  false_alarms=0 data=9648858
rsync[7852] (sender) heap statistics:
  arena:         135168   (bytes from sbrk)
  ordblks:            3   (chunks not in use)
  smblks:             2
  hblks:              0   (chunks from mmap)
  hblkhd:             0   (bytes from mmap)
  usmblks:            0
  fsmblks:           80
  uordblks:       38016   (bytes used)
  fordblks:       97152   (bytes free)
  keepcost:       88904   (bytes in releasable chunk)

Number of files: 4
Number of files transferred: 3
Total file size: 9648858 bytes
Total transferred file size: 9648858 bytes
Literal data: 9648858 bytes
Matched data: 0 bytes
File list size: 109
Total bytes written: 9650279
Total bytes read: 68

wrote 9650279 bytes  read 68 bytes  919080.67 bytes/sec
total size is 9648858  speedup is 1.00
```

As you can see it shows the files transferred, the number of files transferred, the amount of data transferred, and other data relating to the amount and speed of the replication process.

Ongoing replication
Ok, the first replication process has finished lets see how the process handles any changes. I made some changes to the *list.txt* file (note file size change)..

192.168.10.80	ls -l total 9448 -rw-r--r-- 1 root root 28 Jan 8 00:02 list.txt -rw-r--r-- 1 root root 2037760 Jan 7 22:43 nc.tar -rwxr-xr-x 1 root root 7611079 Jan 7 22:44 webmin-1.110-1.noarch.rpm

And lets run the process again and see what we get..

```
rsync -arpogtvv --stats --delete -e ssh /admin/backup/ 192.168.10.13:/admin/backup/
opening connection using ssh 192.168.10.13 rsync --server -vvlogDtpr --delete . /admin/
backup/
root@192.168.10.13's password:
building file list ...
expand file_list to 4000 bytes, did move
done
deleting in .
delta-transmission enabled
./
list.txt
nc.tar is uptodate
webmin-1.110-1.noarch.rpm is uptodate
total: matches=0  tag_hits=0  false_alarms=0 data=28
rsync[7866] (sender) heap statistics:
  arena:         135168   (bytes from sbrk)
  ordblks:            3   (chunks not in use)
  smblks:             5
  hblks:              1   (chunks from mmap)
  hblkhd:        266240   (bytes from mmap)
  usmblks:            0
  fsmblks:          168
  uordblks:       38016   (bytes used)
  fordblks:       97152   (bytes free)
  keepcost:       88904   (bytes in releasable chunk)
```

```
Number of files: 4
Number of files transferred: 1
Total file size: 9648867 bytes
Total transferred file size: 28 bytes
Literal data: 28 bytes
Matched data: 0 bytes
File list size: 109
Total bytes written: 193
Total bytes read: 42

wrote 193 bytes  read 42 bytes  52.22 bytes/sec
total size is 9648867  speedup is 41059.01
```

Here you can see that it recognizes which files are up to date, and then only copies those which aren't. This we can see by the "*number of files transferred*" and the "*total bytes written*" statistics. This will greatly speed up any replication process.

But is it secure?
Now lets see what packets an eavesdropper would see looking in on this particular network conversation..

```
tcpdump -i eth1 "src 192.168.10.80 and dst 192.168.10.13" -X -q -t -s 65535
tcpdump: listening on eth1
192.168.10.80.44478 > 192.168.10.13.ssh: tcp 144 (DF)
0x0000   4500 00c4 c208 4000 4006 e27d c0a8 0a50        E.....@.@..}...P
0x0010   c0a8 0a0d adbe 0016 c9c1 e9ac 1aa4 2c10        ..............,.
0x0020   8018 2580 616b 0000 0101 080a 058b 25b4        ..%.ak........%.
0x0030   0719 2df6 b914 1e60 4d05 0d4f ba0c a077        ..-....`M..O...w
0x0040   a66a 1a72 116f d450 aca0 fe46 c38e 9e0c        .j.r.o.P...F....
0x0050   b452 70bd 7f8d d1e9 2e12 41c8 7143 0638        .Rp.......A.qC.8
0x0060   ac96 b0fd cf8e baba cdfe 20f2 7e68 7732        ............~hw2
0x0070   dbd9 c81b 1b91 09ab d4da 8069 4a74 4722        ...........iJtG"
0x0080   9f19 34e9 e304 bd24 f226 57c4 d251 d0ab        ..4....$.&W..Q..
0x0090   7335 777d eb7c 5c2b f9ca 51ad b698 a6d8        s5w}.|\+..Q.....
0x00a0   a05a ca47 7e8c 313a af52 bfc8 21cf acf7        .Z.G~.1:.R..!...
0x00b0   cdc2 ac86 1d6f 2d1e ece8 90d3 0221 2df0        .....o-......!-.
0x00c0   07d5 2866                                      ..(f
192.168.10.80.44478 > 192.168.10.13.ssh: tcp 0 (DF)
0x0000   4500 0034 c209 4000 4006 e30c c0a8 0a50        E..4..@.@......P
0x0010   c0a8 0a0d adbe 0016 c9c1 ea3c 1aa4 2c30        ...........<..,0
0x0020   8010 2580 b1b8 0000 0101 080a 058b 25b5        ..%...........%.
0x0030   0719 2ed6                                      ....
192.168.10.80.44478 > 192.168.10.13.ssh: tcp 64 (DF)
0x0000   4500 0074 c20a 4000 4006 e2cb c0a8 0a50        E..t..@.@......P
0x0010   c0a8 0a0d adbe 0016 c9c1 ea3c 1aa4 2c30        ...........<..,0
0x0020   8018 2580 6eb6 0000 0101 080a 058b 25b5        ..%.n.........%.
0x0030   0719 2ed6 be5d 5c15 7edc cfe6 f53e cc0b        .....]\.~....>..
0x0040   ad1c 3de4 d1dc 3c3b 3e74 5c4a 1c5b b288        ..=...<;>t\J.[..
0x0050   c6ce 4ced 3ed6 630d b363 be22 11b5 9b52        ..L.>.c..c."...R
0x0060   3e20 03e3 c043 b2f6 ce94 12e7 9b1b 06a0        >....C..........
0x0070   9703 11d7                                      ....
192.168.10.80.44478 > 192.168.10.13.ssh: tcp 96 (DF) [tos 0x8]
0x0000   4508 0094 c20b 4000 4006 e2a2 c0a8 0a50        E.....@.@......P
0x0010   c0a8 0a0d adbe 0016 c9c1 ea7c 1aa4 2c60        ...........|..,`
0x0020   8018 2580 9669 0000 0101 080a 058b 25b6        ..%..i........%.
0x0030   0719 2ed6 9668 b478 5e29 1482 ada9 8c87        .....h.x^).....
0x0040   5531 6903 1206 f63b 7adf 31de 1ea6 c599        U1i....;z.1.....
0x0050   9c08 d01e 5bbd fe6f 17c5 2369 5a5a 7d2f        ....[..o..#iZZ}/
0x0060   7c2e 432b c330 12bc 2c1a f2b7 4944 09c2        |.C+.0..,...ID..
0x0070   d780 9af0 7125 573b ba17 907f 32f2 0ded        ....q%W;....2...
0x0080   809b c429 52e8 edf9 6a39 ed83 0d8e c449        ...)R...j9.....I
0x0090   f5af 7fe2                                      ....
```

Not much help at all is it? We can see that using *ssh* as our *rsync* replication shell has done it's job, all the traffic has been encrypted - always a good thing.

Batch script

Now that we know the process works and is secure, here is a sample little shell script you can use (as always with scripts I make no claim that they are complete and perfect works of art, please feel free to change them to suit you) ...

```
#setup variables
TGT=192.168.10.13
SRC=/admin/backup
DST=/admin/backup
RSYNC=/usr/bin/rsync
CNF=/admin/conf
RPT=report.sync

#setup report file
rm -rf $CNF/$RPT
touch $CNF/$RPT
DT1=`date`
echo $DT1 >> $CNF/$RPT
echo "~~~~~~~~~~~~~~~~~~~~~~~~~" >> $CNF/$RPT
echo "SYNCING $SRC to $TGT $DST" >> $CNF/$RPT
echo "" >> $CNF/$RPT

#do the actual sync
$RSYNC -arpogtv --stats --delete -e ssh $SRC $TGT:$DST >> $CNF/$RPT

#log to syslog
logger "===sync.srv: Sync of $SRC to $TGT $DST finished==="

#close off report file
echo "" >> $CNF/$RPT
echo "~~~~~~~~~~~~~~~~~~~~~~~~~" >> $CNF/$RPT
DT2=`date`
echo $DT2 >> $CNF/$RPT
```

Final words

Used this way *rsync* is a very useful tool, but as always there are many more options you can set depending on your environment. For example, using the *--bwlimit* option allows you to specify the bandwidth that the replication process will use, this prevent the process from becoming a bandwidth hog, so go through the different switches to find the optimum use for you. Also one word of caution, replication is not backup, do not use replication in place of a good backup strategy. Anyway, as always have fun and learn.