The Answer Girl SHOVELING DATA

PATRICIA JUNG

The fact that the world of everyday computing, even under Linux, is often good for surprises is a bit of a truism: Time and again things don't work, or not as they are supposed to. Linux Magazine's Answer Girl shows you how to deal elegantly with such little problems.

__$HOME:__ The home directory of the respective user is stored in the Environment variable HOME. With a $ before the variable name, you can reach its content. So echo $HOME outputs the home directory of the enquiring user on the command line.
__Backup Medium:__ data carrier reserved for the recording of backup data; on a large scale this is usually magnetic tapes and hard disks.
__Rotation:__ To face the risk of a total failure of the backup medium with equanimity, you should if at all possible use a different medium for each backup run. But since this is highly impractical (and data still becomes obsolete anyway at some point), you should use a number of media in rotation, such as, Monday is always tape 1, ..., Sunday is always tape 7.

You have probably heard more than enough of the well-meaning litany about making a backup. At work or at university there may be some justification for leaving responsibility for this tedious activity to system administrators, but what happens to your data at home?

A tape drive is pretty rare at home, a backup on CD requires a CD burner, and if there isn't a blank in the drive at all times, don't even think about automation. Data backup on diskette? You might do that with the letters from the Inland Revenue, but hardly with your 100-page thesis and the exchange of emails with past loves, which has by now grown into several MBs.

Storage strategies

With the current size of hard disks, you can certainly spare the room for a dedicated partition and use it exclusively as backup space. Bad news if the hard disk goes off to the great cyber hunting grounds in the sky, but better than nothing at all.

Better yet is a second disk – even with the six-year-old GB from the cast-off computer you can go a long way. As long as the computer does not get stolen or go up in flames, this is not bad at all.

But by no means should you underestimate the (safer) alternative, of not mothballing the old computer in the first place, but turning it into your own personal backup cupboard. Of course this could also be a notebook, on which data worth preserving is always kept in a second copy. To do this will not need more than a little LAN.

And yet, thanks to flat-rates, ADSL etc., even an account at college or the Internet computer belonging to your partner is a suitable storage place for selected data. Those not wanting to back up their entire 2GB installation but wishing to stick to hand-optimised configuration files and the best of **$HOME**, can presumably make a backup via ISDN or modem. In case of doubt there is still an update to SuSE or Red Hat waiting after the next disk crash.

In the face of such heretical statements, any conscientious system administrator will of course scream blue murder, but hand on heart: You have still not got to grips with any proper backup software have you? Even if you have, this is not backing up your home computer anyway. If you are one of the shining exceptions the question arises, when did you last check whether your backup could actually be restored?

Backup or data reconciliation?

So what we are looking for is an alternative which may not be quite so secure, but on the other hand is easier to manage. Regular backups are usually performed as so-called *incremental backups*. This means that once (or better, at regular intervals) a complete security copy of all data is made and between two full backups, only the differences with respect to the previous version are saved in each case.

Plus, the **backup media** is **rotated**, so if later on something breaks or otherwise turns out to be unusable, you will hopefully be able to fall back on the next oldest backup copy.

If this is applied to our home data, this is of course also the ideal situation but for rotation, several disks or even computers would be necessary for our impossible demand for backup on hard disk. The keyword 'incremental', on the other hand, certainly has its attraction for us – after all, we don't want to back up the data from new every time when it hasn't even changed.

Anyone wanting to have files ready in various processing versions cannot, however, solve this problem using a incremental backup. They would be better off using a version control software such as *cvs*, so that they can settle for a situation where the target system contains precisely the data which was in the source directories before the data reconciliation – no more, but no less either.

So what we want is a simple *mirroring* of the data, preferably via the network and, if at all possible, in such a way that the data (and especially the password) are encrypted. For a simple restoration of the data to be possible, there should not be any accumulation of files in the target directories, which have already been deleted from the source directories. This means that before each data reconciliation we must be sure that all previous deletions were correct – that is the price to be paid for not rotating backup media.

Your choice of which directories to back up should be determined by the following criteria: The capacity of the target system, the form of network connection between the two computers and your personal evaluation of which data is actually worth backing up

A question of software

If there is an FTP server running on the target system, you can of course use it, but this means transmitting password and data in clear text over the network. Also, FTP-client programs aren't usually capable of transferring the most recent data *only*, or automatically deleting data that no longer exists in the source directories. If you have to use FTP as the method of transfer, it's best to stick to *mirror software*, which provides proper backup programs.

The *Secure Copy* program *scp*, which comes with the Linux version of the **SecureShell** or its open-source pendant OpenSSH, is certainly suitable for this. Here, all the data travels over the network encrypted. A secure shell server, the daemon *sshd*, should nevertheless be on every Internet computer on which you do not wish to work only from the local **console**. Nevertheless, some of the criticism of FTP clients also applies to *scp*: It should not be used for a data *reconciliation*.

Anyone who has been involved with Unix for a while may recall that the unencrypted pendant of *scp* is called *rcp*. Many people were irritated by the fact that this cannot perform a data reconciliation, and these included Paul Mackeras and Andrew Tridgell, the latter being better known from **Samba**. And because their *rcp* substitute (called *rsync*) can also perform an encrypted data reconciliation via *ssh*, it's worth a trip to *http://rsync.samba.org/rsync/download.html*, if the distribution does not come with a suitable packet.

Decrypting

A *man rsync* intervenes initially, so that in the *SYNOPSIS* chapter all combinations of data transfer options are presented schematically:

```
rsync [OPTION]... SRC [SRC]...
[USER@]HOST:DEST
rsync [OPTION]... [USER@]HOST:SRC DEST
rsync [OPTION]... SRC [SRC]... DEST
rsync [OPTION]... [USER@]HOST::SRC [DEST]
rsync [OPTION]... SRC [SRC]...
[USER@]HOST::DEST
rsync [OPTION]...
rsync://[USER@]HOST[:PORT]/SRC [DEST]
```

As usual in the case of the *Backus-Naur Form notation* used in manpages, options in square brackets can be left out. The three dots do not

Client: *'customer', making use of the services of a server. The term is used to refer both to the computer on which a client program is running, as well as for this program itself. This means a computer can be both client and server at the same time.*

SecureShell: *A safe replacement for the traditional Remote-Login or r-services Telnet and RSH (Remote Shell). A remote login, such as logging onto a distant computer, makes it possible, while working on a local computer, to access a computer connected via the network as if you were sitting right in front of it. To do this, one starts a remote login client on the local computer (such as telnet, rsh or ssh), which converses with the remote server (telnetd, rshd or sshd). With a secure shell connection, unlike Telnet or the r-services, all data is transmitted encrypted.*

Console: *The unit forming part of a computer, consisting of (local) screen and keyboard.*

Samba: *Windows computers can allow mutual access to their files and/or printers. The exchange of data is transacted according to the rules of the Server Message Block network protocol, where messages travel back and forth in blocks between server and client computers. Samba is software that implements this protocol and thereby also gives Linux and other Unix computers the option of allowing such SMB accesses and/or access to approved resources.*

■

__IP Address:__ Unique identity of a computer on the Internet – either as a combination of numbers. In the current, commonest Version 4 of the Internet Protocol a maximum of four, three-digit numbers separated by dots or as text, consisting of domain and computer name, such as www.linux-magazine.co.uk. The conversion of numerical and textual IP addresses is taken over by Nameservers, also known as DNS (Domain Name System) servers.

__Port:__ If all planes/trains arriving at roughly the same time at a large airport or station were to go to the same gate/platform, there would be rather a lot of collisions. A computer offering various services (server) is confronted with a similarly precarious position with respect to network traffic. This is why every server process (daemon) eavesdrops at a different 'gate/platform' – the port. When a daemon listens out for a port which is reserved for its service, a Wellknown port, the client does not normally need to state a port number. But if the server uses a different port (a Web server using 8080 instead of 80), the client must be told of this explicitly.

If it is written in the GENERAL section that both lines with the double colon (::) equally require an rsync daemon, it is only the first three lines which are of interest to us:

1. Copy local files/directories into a directory on a remote computer.

2. Pack copies of remote data into a local directory.

3. Mirror local data in a different local directory

__~:__ The tilde is an abbreviation of the shell for the home directory of the present user. If there is a username after the ~, this means the home directory of this user.

exactly correspond to a scientifically precise nomenclature, but it does make clear what the authors want to say: There can be more details of the type just described written here (for example additional options).

Just as easy as decrypting the *[OPTION]...* placeholder as "any number of the options listed below in the section *OPTIONS SUMMARY*", is the demystification of *[USER@]* name and a following @) and *HOST*. These are the optional specification of a user and the numeric or textual **IP-Address** of the remote computer respectively.

With our expectations of file and directory transfer, the only way to interpret *SRC* and *DEST* is as *Source* and *Dest*ination files/directories. Since we wish to transfer our data via the SecureShell protocol, the last option does not interest us, - that of addressing an *rsync* server on the **Port** *PORT*, so that we can forget the last line.

Are you local?

We'll begin with the last and simplest case: The directory *~/article* is to be copied as backup onto another partition mounted under */mnt/backup*.

```
[trish@lillegroenn ~]$ rsync article /mnt/backup
skipping directory /home/trish/article/.
```

That was not exactly a rush of copying: There is not a single file in the destination directory */mnt/backup*. Now we must look, for the options:

```
Options
[...]
-r, –recursive          recurse into
directories
```

Anyone wanting to copy entire directories together with their content should thus also specify a *-r* or *–recursive* at the same time:

```
[trish@lillegroenn ~]$ rsync -r article /mnt↵
/backup
```

The disk noise does indicate that something is happening, but what?

```
write failed on
article/LM/LM0501/ootb/gramofile-3.html : No↵
such file or directory
unexpected EOF in read_timeout
unexpected EOF in read_timeout
```

A fast *df* (*d*isk *f*ree) confirms our fears:

```
Filesystem    1k-blocks      Used Avail Use% ↵
Mounted on
/dev/hda2       643959    610690     5 100% /↵
mnt/backup
```

The partition is full! So we first delete the failed backup with **rm -rf** */mnt/backup/article* completely and *r*ecursively.

The thing to do now is to find out using *du* where the miscreants are hiding. To prevent the thousand sub and sub-sub-directories rushing right past us, we shall limit ourselves to the first two directory levels under *~/article*:

```
[trish@lillegroenn ~]$ du –max-depth=2 article
[...]
1924    article/LM/LM0501
51      article/LM/LM0601
73270   article/LM
[...]
84049   article/designer/qt-designer2
234     article/designer/qt-designer1
100112  article/designer
[...]
```

The numbers in the first column, the size of the directory contents in Kbytes, are still extremely unclear. The miscreant is quite certainly more than 1MB in size, and luckily *du* has the option *-m*, with which the size details are stated in rounded whole MB.

Then there's a whole series of zeroes for the directories that are smaller than 1MB. To see only the larger directories, we set *awk* to work:

```
[trish@lillegroenn ~]$ du -m –max-depth=2 ar↵
ticle | awk '$1 > 1'
[...]
2       article/LM/LM0501
72      article/LM
[...]
82      article/designer/qt-designer2
98      article/designer
[...]
```

awk now filters out all *du* output lines in which the first column (*$1*) is greater than *1*, and does not display the rest at all.

In this way we have detected that the miscreant is *~/article/designer/qt-designer2* and as this directory contains only test software, we can also do without the backup of it. With the *-exclude* flag we now tell *rsync* that it should ignore all files containing a *qt-designer2* in the **path** or file name. But this time we are more cautious and do a dry run first with *-n* (*n*ot actually to be executed):

```
[trish@lillegroenn ~]$ rsync -rn --exclude "q↵
t-designer2"  article /mnt/backup
```

The emergency without the *-n* precaution option is causing problems again, though:

```
[trish@lillegroenn ~]$ rsync -r --exclude "qt↵
-designer2"  article /mnt/backup
[...]
skipping non-regular file article/designer/q↵
t-designer1/qt-2.2.3/include/qxml.h
```

A look, using *ls -l*, at the suspect file brings the explanation:

```
lrwxrwxrwx   1 trish    users        17 Dec ↵
21 05:32 article/designer/qt-designer1/qt- 2↵
.2.3/include/qxml.h -> ../src/xml/qxml.h
```

The file in question is a link which was simply not copied with the others. But there is also a remedy for this: with the *rsync* option *-l*, with which symbolic links are retained.

The manpage, in the section *USAGE*, also kindly explains that the archive option *-a* simultaneously copies recursively, retains links and doesn't change attributes, rights, the owner details, or any device files either. Exactly what we want for backup! Quite incidentally, we also learn here about the verbosity option *-v*, which we shall also use from now on in our tests.

There is still one problem: If we don't bear in mind that files deleted in the source directory also disappear from the backup, multiple deletions will at some point fill up the backup partition. Quite apart from that, when a backup is really necessary, it is tedious clearing up all the files which had long since been thrown away, after playing back the data.

The corresponding *rsync* option, which deletes everything at the destination site that no longer exists at the source site, is called *– delete*. So let's make a full backup, then rename a file from *~/article* for test purposes and see what happens:

```
[trish@lillegroenn ~]$ rsync -av --exclude "q↩
t-designer2"  article /mnt/backup
[many files]
article/LM/LM0601/Answergirl_0601.html
[many files]
[trish@lillegroenn ~]$ mv article/LM/LM060↩
1/Answergirl_0601.html !#:1_new
mv article/LM/LM0601/Answergirl_0601.html ar↩
ticle/LM/LM0601/Answergirl_0601.html_new
[trish@lillegroenn ~]$ rsync -av --delete --ex↩
clude "qt-designer2"  article /mnt/backup
building file list ... done
article/LM/LM0601/
deleting article/LM/LM0601/Answergirl_0601.html
article/LM/LM0601/Answergirl_0601.html_new
article/LM/LM0601/
wrote 43868 bytes  read 32 bytes  29266.67 by↩
tes/sec
total size is 26953280  speedup is 613.97
```

rsync dutifully reports that it is *deleting* the file *Answergirl_0601.html* which no longer exists in *~/article/LM/LM0601* in */mnt/backup/article/LM/LM0601* too and instead is creating the new file *Answergirl_0601.html_new*.

With *!#* we are telling the **Bash** that it should instead implement everything which has been on this command line until now (*mv article/LM/LM0601/Answergirl_0601.html*). Thanks to *:1* we are somewhat more selective and tell the shell to restrict itself to argument number 1 (the second argument *article/LM/LM0601/Answergirl_0601.html*).

Anyone who likes to play safe and wants to retain a safety copy of all amended files (thus even the deleted ones) in the backup directory, will presumably become familiar

with the *rsync* option *-b*. This is by no means a substitute for version control, but could be of interest to more than just the nervous. By default, the backup files are given a tilde after a file name.

Off in the distance

We do not really need much more if we are limiting ourselves to the local mirroring of data. But it is always safer to have a copy on a different computer. If we recall the synopsis, this was also very easily realised by *rsync*: If the usernames are different on the source and destination computers, the latter must be stated with a following @ before the address of the remote computer. There is also a colon at the end, after which the destination directory can be written - or nothing, if we are settling for the remote home directory:

```
[trish@lillegroenn ~]$ rsync -av --delete ar↩
ticle pjung@backup.linux-magazine.co.uk:
```

Since there are hopefully no **r-services** running on the remote computer, there ought to be an error message. We're better off going via SecureShell at this point, provided there is a *sshd* running on *backup.linux-magazine.co.uk*. To get *rsync* to transfer via SecureShell, there are the options *-e* ("execute") or *–rsh* ("substitute for *rsh*"). The former wants the *ssh* command after a space, the latter wants an equal- sign (*–rsh=ssh*):

```
[trish@lillegroenn ~]$ rsync -av --delete -e s↩
sh article pjung@backup.linux-magazine.co.uk:
```

If your *ssh* command does not lie in the search path, you must of course state the full path, *-e /usr/local/bin/ssh*. So you don't want to make the *article* directory on the destination computer directly underneath *pjung's* home directory? Then we must also explicitly specify the destination parent directory, such as:

```
[trish@lillegroenn ~]$ rsync -av --delete -↩
e ssh article pjung@backup.linux-magazine.c↩
o.uk:~/backup
```

The *USAGE* manpage section revealed, if you can recall, that the data is transferred compressed with *-z*. This certainly plays a role now as our data is going via the network, which is why we are adding this option, before actually pressing the Enter key:

```
[trish@lillegroenn ~]$ rsync -avz --delete -↩
e ssh article pjung@backup.linux-magazine.c↩
o.uk:~/backup
pjung@backup.linux-magazine.co.uk's password:
[enter password]
building file list ... done
```

Better with script

Repeatedly typing in this whole rigmarole – well, we're much too lazy to do that. Anyone wanting to back up several directories or even individual files (such as the bookmarks of a Web browser) will be longing for a little script, which – once written – can if possible even be processed automatically by a **Cronjob**.

It is best if we write the files to be backed up as a list separated by spaces in a variable named

.rm -rf: *One of the most notorious Unix commands of all: It deletes, without challenge, (-f stands for force) an argument directory together with all subdirectories. Before you set this command off, then, you should be really sure that you have not included any typing errors: An rm -rf /mnt/backup leaves just an empty /mnt behind, if backup was previously the only directory entry in /mnt.*

Path: *The sequence of directories via which one must go if one wants to reach a certain file in the file tree.*

Bash: *The 'Bourne Again Shell' is used by most distributions as the standard command line interface. A Shell accepts user inputs and transforms them so they turn into orders (program commands) for the kernel.*

r-services: *See explanation on SecureShell in this article.*

Cronjob: *Task in a Cron table, which is executed by the Cron daemons at a specified time repeatedly and automatically without any action on the part of the user. cf. the manpages on cron(8), crontab(1) and crontab(5).*

■

rsync 2.4.6

BACKUPFILES, while remote user name, @, the address of the remote computer, the colon and the destination directory are easy to amend in *BACKUPTARGET*. For the script equivalent to the command

```
[trish@lillegroenn ~]$ rsync -avz --delete -↩
e ssh article .netscape/bookmarks.html pju↩
ng@backup.linux-magazine.co.uk:~/backup
```

the variable contents therefore look as in Listing 1.

But wait – why is the *.netscape* subdirectory now missing in *backup.linux-magazine.co.uk* in the *~/backup*-directory, so that *bookmarks.html* is suddenly present as *~/backup/bookmarks.html*? Because we, as the *rsync* manpage shows, forgot the option *-R* (*r*elative), which makes sure that on the destination computer exactly the same relative paths are installed as on the source computer.

No password

If, for example, one wishes to automate the data reconciliation using a Cronjob (cf. Crontables, LM Issue 6 p.108ff.), entering a password turns into a problem. It can be resolved, even if security fanatics might need to close one eye.

The keyword is *Public Key Cryptography*: One has a pair of keys, of which one of the keys is kept secret and the other is publicly distributed. Authentication is only possible when both secret and public key come together. As we can see from the manpage on *ssh*, our chosen method of transfer supports this.

What we have to do first is to generate the key for the computer executing the backup script. We could almost have guessed it: The command for this is called *ssh-keygen* ("*ssh-key gen*eration" - creating the *ssh* key).

```
[trish@lillegroenn ~]$ ssh-keygen
Generating RSA keys:  .................oooo↩
ooO................oooooooO
Key generation complete.
Enter file in which to save the key (/home/tr↩
ish/.ssh/identity):  [Enter]
Enter passphrase (empty for no passphrase): ↩
[Enter]
Enter same passphrase again:  [Enter]
Your identification has been saved in /home/↩
trish/.ssh/identity.
Your public key has been saved in /home/trish↩
/.ssh/identity.pub.
The key fingerprint is:
f7:68:22:9f:a3:be:37:7c:7f:92:c2:fb:a1:86:ff↩
:fe trish@lillegroenn.troll.no
```

Anyone wanting to save their secret key in the suggested file *~/.ssh/identity*, simply confirms with just the *Enter* key, otherwise a file name, preferably with path, is necessary.

It gets critical when it comes to the request for the password: Normally we would set one to protect the private key, but then we would have to enter one again – an infinite circle. That's why this time we are going to swallow the bitter pill and again type only *Enter*. Also, at the last request to repeat the (now blank) password it is still appropriate to enter nothing but *Enter*. Anyone finding the no-password key unsettling can still increase security by frequently generating and distributing a new key.

We shall now take the *public* key (saved with the ending *.pub*) and transfer it to the backup computer via SecureShell, of course (thus with *scp* or by copy & paste, while logged onto *ssh*). It must in any case be entered into the file there *~/.ssh/authorized_keys*, like this:

```
[trish@lillegroenn ~]$ cat ~/.ssh/identit↩
y.pub | ssh -v pjung@backup.linux-magazine.c↩
o.uk cat - > ~/.ssh/authorized_keys
```

This fiddly procedure, instead of a simple *scp ~/.ssh/identity.pub pjung@backup.linux-magazine.co.uk:~/.ssh/authorized_keys* is necessary when *~/.ssh/authorized_keys* is already accommodating other keys on *backup.linux-magazine.co.uk*, too. As the result of the double > what we achieve is that the standard input which is output with the second *cat* of *ssh* (symbolised by a -) is attached to the end of *~/.ssh/authorized_ keys*.

Where does this entry for *ssh* come from, which passes the latter on to the remote command to be executed *cat - > ~/.ssh/authorized_keys*? The pipe | is responsible for this, which shoves the output of *cat ~/.ssh/identity.pub* into the *ssh* command.

All that's left now is to test whether the script actually functions without a password. If so, there is no further obstacle to a backup Cronjob. ■

Listing 1: Backup script

```
#!/bin/sh
# files and directories to be backed up, starting
# from the home directory
BACKUPFILES="article .netscape/bookmarks.html"
# Backup target
BACKUPTARGET="pjung@backup.linux-magazine.co.uk:~/backup"
cd  # Change to home directory
rsync -e ssh -aRvz –delete $BACKUPDIRS $BACKUPTARGET
```

Replace the italic details, save the file, and use chmod u+x to give it the necessary execution rights for yourself. Then the script can be executed by calling up its name (if necessary with path details).

Reciprocal data reconciliation

Notebook owners often get annoyed about inconsistencies in the data stored on desktop computer and notebook. The solution sounds simple: A script as in Listing 1 is installed on both computers, and depending on which computer was lasted worked on, the data on the other computer is updated ... and in the worst case, overwrites a more recent version on the destination system with the old one. Here the rsync option -u (update only) can help. This ensures that files with a more recent time stamp on the destination system than on the source system are not overwritten. One important point here: the computer time on the two systems absolutely must be synchronised.