Back Up to Another System

Saturday, February 24, 2018 2:23 PM

The following information will show two ways to perform a synchronized backup from one system to another.

The first method is invoked manually. The following example assumes there is a directory (could be a mounted disk) named **disk1** on another system with a sub-directory named **sys2backup**. The address of the system containing these directories is 192.168.1.101. We have a directory on our current system named /**disk0** (could be another mounted disk) and we are going to synchronize (backup) this directory with the other system.

From a command prompt enter:

rsync -avz --delete /disk0 pi@192.168.10.12:/disk1/rpiXX

A couple of things to note about this method:

1. The ***--delete*** command tells **rsync** that if there are files on the target that don't exist on the source, delete them. This creates a backup that is **synchronized** with the source. If you effectively want an **accumulation** of anything that ever existed on the source, remove this option.

2. The ***-avz*** options tell **rsync** to:
   a. a - operate in **archive** mode (this effectively combines 7 **rsync** options, including ***r*** - recursive, to cause **rsync** to recurse or process sub-directories).
   b. v - use **verbose** mode which causes more information to be displayed during the process.
   c. z - causes data to be compressed during the transfer.

3. When you execute this command, **rsync** will ask you for credentials (ex: password) for the other system.

The second method is to use the same command but implement it with **cron** to run automatically at scheduled day(s) and time(s). Something to remember is that when the command is run it will ask you for credentials. Obviously, this can be problematic if you aren't connected to the source system when the command executes.

One way to circumvent this issue is to generate a **key** to be used between the source and target systems. This effectively creates a way for them to "recognize" each other.

From a command prompt, enter: ***ssh-keygen -t rsa***

1. When you are asked for a password, **do not enter one**. Just use the enter key when asked. The command will show where the files have been created, the key fingerprint and what is called a "randomart image" of the key.

2. The default to save the key (**identification**) for the source system is: ***/home/pi/.ssh/id_rsa***

3. The default to save the key for the target system (**public key**) is: ***/home/pi/.ssh/id_rsa.pub***

4. The target system needs a file called **authorized_keys**. In this example, that file should be located in the following directory: ***/home/pi/.ssh/authorized_keys***

5. **IMPORTANT:** If **authorized_keys** does not exist on the target machine, move the file **rsa.pub** to the **/home/pi.ssh** directory and execute: ***cat id_rsa.pub > authorized_keys***.

   IF authorized_keys already exists, replace the **>** with **>>** in the previous command. Instead of creating the file this will **concatenate** the new key to any others that may have previously existed.

You can now test your shared key setup by executing the previously discussed manual version of the backup from the command line. **rsync** should run without asking for any credentials.

NOTE: This setup assumes the crontab entries are being established by the user **pi** (or another user whose name is being used in the paths referenced above). If this is not the case, the crontab entry needs to be changed. For example, if the paths referenced **pi**, but the crontab entry was made by a **sudo** command, the crontab is stored in a different file. When executed, the system would not look for the shared key in **pi's** .ssh directory. Therefore, you need to include its location in the command itself. The example we used above would then look like:

rsync -avz --delete "ssh -i /home/pi/.ssh/id_rsa" /disk0 pi@192.168.1.101:/disk1/sys2backup

There is also a system-wide version of cron. For more information on this, see the manual page at:
`cron(8)` manpage