

Incremental Backups

Sheetal Joseph

March 28,2005

Abstract

Grr... I am in extremely foul mood today. One of the major server went down today. It is a near disaster. But luckily, I had a backup plan. Yes, I had taken backup of data in some other place, and phew! I breath easy. It was not the case, couple of months ago, when I was stuck up with a server crash issue, with none so backup available. Total nervous breakdown situation, it was. That memory still haunts me.

Intially I used to be a bit reluctant in taking backups. I kept putting it off because I felt I didn't want to consume huge amount of disk space, by keeping redundant copies of key directories. Keeping track of these data, was also another major headache.

Then, my pal Appu refered to me to a backup scheme called `rsnapshot`, which is build over `rsync`. It is an incremental backup scheme, which helps in keeping the backup size to almost the same size of the original data. This document discusses the basic idea behind incremental backup, and then further proceeds onto explain how to automate your backups using `rsnapshot`

Contents

1	Problem statement	1
2	Using rsync to make a backup	1
2.1	Basics	2
2.2	Trailing Slashes Do Matter...Sometimes	2
2.3	Using the <code>-delete</code> flag	2
3	Incremental backups with rsync	2
3.1	Review of hard links	2
3.2	Using <code>cp -al</code>	3
3.3	Putting it all together	3
4	rsnapshot	3
5	Cron is your friend	3
6	How it works	4
7	Making the Backup as readonly as possible	4
7.1	A possible solution: using NFS/Samba on localhost	5
8	Final few words	5
9	Reference	5

1 Problem statement

In a nutshell here's the problem we're solving

1. We have a bunch of users on a web server called *mordor*. Server runs Fedora Core 2, Kernel 2.6.9.
2. We have a second FC2 machine, *rohan*, with good IP connectivity to *mordor*.
3. Backups are desired every hour during the main part of each day, once a day for a week, and once a week for a month.
4. The backups should be incremental - minimising storage requirements by only keeping 'diffs' from one backup to the next.
5. Users on *mordor* should be able to access their backups - going back hours, days, or weeks - without system administrator intervention.

2 Using rsync to make a backup

The `rsync` utility is a very well-known piece of GPL'd software, written originally by Andrew Tridgell and Paul Mackerras. If you have a common Linux or UNIX variant, then you probably already have it installed; if not, you can download the source code from <http://rsync.samba.org>. Rsync's specialty is efficiently synchronizing file trees across a network, but it works fine on a single machine too.

2.1 Basics

Suppose you have a directory called `source`, and you want to back it up into the directory `destination`. To accomplish that, you'd use:

```
rsync -a source/ destination/ (Note: It is a good practice to use the -v (verbose) flag too, so that rsync tells us what it's doing).
```

This command is equivalent to: `cp -a source/. destination/` except that it's much more efficient if there are only a few differences.

Just to whet your appetite, here's a way to do the same thing as in the example above, but with `destination` on a remote machine, over a secure shell:

```
rsync -a -e ssh source/ username@remotemachine.com:/path/to/destination/
```

2.2 Trailing Slashes Do Matter...Sometimes

This isn't really an article about `rsync`, but I would like to take a momentary detour to clarify one potentially confusing detail about its use. You may be accustomed to commands that don't care about trailing slashes. For example, if `a` and `b` are two directories, then `cp -a a b` is equivalent to `cp -a a/ b/`. However, `rsync` does care about the trailing slash, but only on the source argument. For example, let `a` and `b` be two directories, with the file `foo` initially inside directory `a`. Then this command:

```
rsync -a a b produces b/a/foo,
```

whereas this command: `rsync -a a/ b` produces `b/foo`. The presence or absence of a trailing slash on the destination argument (`b`, in this case) has no effect.

2.3 Using the `--delete` flag

If a file was originally in both `source/` and `destination/` (from an earlier `rsync`, for example), and you delete it from `source/`, you probably want it to be deleted from `destination/` on the next `rsync`. However, the default behavior is to leave the copy at `destination/` in place. Assuming you want `rsync` to delete any file from `destination/` that is not in `source/`, you'll need to use the `--delete` flag:

```
rsync -a --delete source/ destination/
```

3 Incremental backups with rsync

Since making a full copy of a large filesystem can be a time-consuming and expensive process, it is common to make full backups only once a week or once a month, and store only changes on the other days. These are called *incremental* backups

3.1 Review of hard links

A hard link is essentially a label or name assigned to a file. Conventionally, we think of a file as consisting of a set of information that has a single name. However, it is possible to create a number of different names that all refer to the same contents. Commands executed upon any of these different names will then operate upon the same file contents.

To make a hard link to a pre-existing file, enter:

```
ln a b
```

This will create a new item in your working directory, `b`, which is linked to the contents of `a`. The new link will show up along with the rest of your filenames when you list them using the `ls` command. This new link is not a separate copy of the old file, but rather a different name for exactly the same file contents as the old file. Consequently, any changes you make to `a` will be visible in `b`.

So `ln a b` is roughly equivalent to `cp a b`, but there are several important differences:

- The contents of the file are only stored once, so you don't use twice the space.
- If you change `a`, you're changing `b`, and vice-versa.
- If you change the permissions or ownership of `a`, you're changing those of `b` as well, and vice-versa.
- If you overwrite `a` by copying a third file on top of it, you will also overwrite `b`, unless you tell `cp` to unlink before overwriting. You do this by running `cp` with the `--remove-destination` flag. Notice that `rsync` always unlinks before overwriting!!.

But this raises an interesting question. What happens if you `rm` one of the links? The answer is that `rm` is a bit of a misnomer; it doesn't really remove a file, it just removes that one link to it. A file's contents aren't truly removed until the number of links to it reaches zero. In a moment, we're going to make use of that fact, but first, here's a word about `cp`.

3.2 Using `cp -al`

In the previous section, it was mentioned that hard-linking a file is similar to copying it. It should come as no surprise, then, that the standard GNU coreutils `cp` command comes with a `-l` flag that causes it to create (hard) links instead of copies (it doesn't hard-link directories, though, which is good; you might want to think about why that is). Another handy switch for the `cp` command is `-a` (archive), which causes it to recurse through directories and preserve file owners, timestamps, and access permissions.

Together, the combination `cp -al` makes what appears to be a full copy of a directory tree, but is really just an illusion that takes almost no space. If we restrict operations on the copy to adding or removing (unlinking) files—i.e., never changing one in place—then the illusion of a full copy is complete. To the end-user, the only differences are that the illusion-copy takes almost no disk space and almost no time to generate.

3.3 Putting it all together

We can combine `rsync` and `cp -al` to create what appear to be multiple full backups of a filesystem without taking multiple disks' worth of space. Here's how, in a nutshell:

```
rm -rf backup.3
mv backup.2 backup.3
mv backup.1 backup.2
```

```
cp -al backup.0 backup.1
rsync -a --delete source_directory/ backup.0/
```

If the above commands are run once every day, then `backup.0`, `backup.1`, `backup.2`, and `backup.3` will appear to each be a full backup of `source_directory/` as it appeared today, yesterday, two days ago, and three days ago, respectively—complete, except that permissions and ownerships in old snapshots will get their most recent values. In reality, the extra storage will be equal to the current size of `source_directory/` plus the total size of the changes over the last three days!!!

4 rsnapshot

`rsnapshot` is a filesystem backup utility, written to do exactly what is mentioned above. `rsnapshot` is written in Perl, and depends on `rsync`. `OpenSSH`, `GNU cp`, `GNU du`, and the `BSD logger` program are also recommended, but not required. The latest version of this program is available at <http://www.rsnapshot.com>. Information regarding the installation and configuration of the program is available at the above mentioned web site.

5 Cron is your friend

The first rule for simple, automated tasks under unix is that `cron` is your friend. Under Linux you can, as a regular user, create your own set of cron jobs with `'crontab -e'`. As root you can create a set of system cron jobs (which will run as root or any other user you specify) simply by editing `/etc/crontab`.

So, requirement 3 can be met by three rules in `/etc/crontab` set to handle hourly, daily, and weekly backup snapshots.

We have the shell `rsnapshot` script that takes a single parameter `'hourly'`, `'daily'`, or `'weekly'` to indicate what level of backup is required. Further assume you required: hourly backups at 45 minutes past every hour between 10:45am and 1:45am every day daily backups at 3:15am every day weekly backups at 3:30am every Saturday

The following `/etc/crontab` lines would achieve the stated goals:

```
45    0,1,10-23 * * *    root    /bin/csh /root/bin/rsnapshot hourly
15    3 * * *    root    /bin/csh /root/bin/rsnapshot daily
30    3 * * *    Sat    root    /bin/csh /root/bin/rsnapshot weekly
```

There is no need to do anything more than edit `/etc/crontab` - the cron daemon re-reads `/etc/crontab` every minute to determine what has changed.

6 How it works

We have a snapshot root under which all backups are stored. By default, this is the directory `./snapshots/`. Within this directory, other directories are created for the various intervals that have been defined. In the beginning it will be empty, but once `rsnapshot` has been running for a week, it should look something like this:

```
[root@rohan]# ls -l ./snapshots/
drwxr-xr-x  7 root  root      4096 mar 28 03:15 daily.0
drwxr-xr-x  7 root  root      4096 mar 27 03:15 daily.1
drwxr-xr-x  7 root  root      4096 mar 26 03:15 daily.2
drwxr-xr-x  7 root  root      4096 mar 25 03:15 daily.3
drwxr-xr-x  7 root  root      4096 mar 25 03:15 daily.4
drwxr-xr-x  7 root  root      4096 mar 25 03:15 daily.5
drwxr-xr-x  7 root  root      4096 mar 25 03:15 daily.6
drwxr-xr-x  7 root  root      4096 mar 28 16:45 hourly.0
drwxr-xr-x  7 root  root      4096 mar 28 15:45 hourly.1
drwxr-xr-x  7 root  root      4096 mar 28 14:45 hourly.2
```

```
drwxr-xr-x    7 root    root        4096 mar 28 13:45 hourly.3
drwxr-xr-x    7 root    root        4096 mar 28 12:45 hourly.4
drwxr-xr-x    7 root    root        4096 mar 28 11:45 hourly.5
drwxr-xr-x    7 root    root        4096 mar 28 03:30 weekly.0
```

Inside each of these directories is a full backup of that point in time. The destination directory paths you specified under the `backup` and `backup_script` parameters get stuck directly under these directories.

Each subsequent time `rsnapshot` is run with the `hourly` command, it will rotate the `hourly.X` directories, and then copy the contents of the `hourly.0` directory (using hard links) into `hourly.1`. When `rsnapshot daily` is run, it will rotate all the `daily.X` directories, then copy the contents of `hourly.5` into `daily.0`. `hourly.0` will always contain the most recent snapshot, and `daily.6` will always contain a snapshot from a week ago.

7 Making the Backup as readonly as possible

We want to avoid leaving the snapshot backup directory mounted read-write in a public place. Unfortunately, keeping it mounted read-only the whole time won't work either—the backup process itself needs write access. The ideal situation would be for the backups to be mounted read-only in a public place, but at the same time, read-write in a private directory accessible only by root, such as `/root/snapshot`.

7.1 A possible solution: using NFS/Samba on localhost

Mount the partition where backups are stored somewhere accessible only by root, such as `/root/snapshot`. Then export it, read-only, via NFS, but only to the same machine. That's as simple as adding the following line to `/etc/exports`:

```
/root/snapshot 127.0.0.1(secure,ro,no_root_squash)
```

then start `nfs` from `/etc/rc.d/init.d/`. Finally mount the exported directory, read-only, as `/snapshot`:

```
mount -o ro 127.0.0.1:/root/snapshot /snapshot
```

You can put this entry in `/etc/fstab`, to get the snapshot directory mounted automatically on start up.

And verify that it all worked:

```
mount
...
/dev/hdb1 on /root/snapshot type ext3 (rw)
127.0.0.1:/root/snapshot on /snapshot type nfs (ro,addr=127.0.0.1)
```

At this point, we'll have the desired effect: only root will be able to write to the backup (by accessing it through `/root/snapshot`). Other users will see only the read-only `/snapshot` directory.

Now, all a users have to do to recover old files is go into the `/.snapshots` directory, select the interval they want, and browse through the filesystem until they find the files they are looking for. They can't modify anything in here because NFS will prevent them, but they can copy anything that they had read permission for in the first place. All the regular filesystem permissions are still at work, but the read-only NFS mount prevents any writes from happening.

8 Final few words

I hope, you must have got `rsnapshot` or an equivalent backup policy working by now. A word of advice from me. Don't be lazy, start backing up your data, starting from now. It really can become live saver at times

9 Reference

1. <http://rsync.samba.org>
2. <http://www.rsnapshot.com>
3. http://www.mikerubel.org/computers/rsync_snapshots/