Rsnapshot: A remote filesystem snapshot utility written in perl

Introduction

rsnapshot is an open source filesystem snapshot utility for making backups of local and remote systems, licenced using the GNU General Public Licence and written in perl. Using rsync[1] and hard links, it is possible to keep multiple, full backups instantly available while only taking the time, disk space and network bandwidth of incremental backups. rsnapshot is an excellent example of perl being used as glue.

History

Some years ago, Mike Rubel, frustrated at how hard it is to make backups on a budget, wrote a little shell script[2]. Essentially, his script called rsync to make backups into directories on another filesystem, linked the files in them together, and rotated those backups so that he could easily see that, eg, this particular backup is from three days ago. In 2003, Nathan Rosenquist took Rubel's idea, rewrote it in perl adding several features - such as having an external config file instead of embedding the configuration in variables in the shell script - and released it to the world under the GPL on Sourceforge as rsnapshot. Then in late 2005, when Rosenquist was getting too busy to maintain it, he asked for volunteers to take over. I volunteered.

How it works

Before I talk about how rsnapshot works, I need to talk about hard links and Unix filesystems.

- **Directory entries**

  When most people think of a file, they think of a path such as `/usr/bin/rsnapshot`. That, however, is not a file. It is a directory entry (often abbreviated to `dirent` because that's what the C structure representing them is called). A directory entry has two parts. The first is the filename, such as `usr`, `bin` or `rsnapshot` and the second is an `inode number`, which is effectively a pointer to a chunk of disk space which contains all the rest of the information about a file.

- **Inodes**

  An inode is a complex structure which contains all the information about things like who owns the file, the file's mode, when it was last modified and so on. It also contains structures which you may think of as a pointer telling where on the disk the file is stored.

  Compare this to the much simpler DOS filesystem, where you have directory entries which contain all the meta-information and which point directly to files. The extra level of indirection that Unix filesystems have in the middle is the key to how hard links work.

- ## Hard links

  When checking a filesystem for errors, in both DOS and Unix, it is an error for there to be two pointers to the same file. In DOS this would mean two directory entries for one file - the dreaded "cross-linked files" that `chkdsk` used to grouch about. In Unix it would mean two *inodes* for the same file. But because of the extra pointer de-reference in the middle, it is just fine to have more than one *directory entry* pointing to an inode, and hence to have multiple directory entries - that is, filenames, possibly in completely different directories - leading you to a single file. All these different pointers to the same inode are known as *hard links*.

  You can create hard links using the `ln` command, and you can see them in the filesystem using `ls -l`. The second field on the `ls` output tells you the number of directory entries pointing to the inode (this number of links is a field in the inode itself). `ls -li` will prepend the inode number. And finally for completeness, you can create hard links in C or perl with the `link()` function.

  There are two limitations on hard links. The first is that you can't link to a directory (which is really just a specially structured file in the filesystem, and has a normal inode just like regular files do) because that could create circles in the filesystem which would be Bad. The second is that you can't hard link across filesystems. That's because both the pointers involved - that in the dirent and that in the inode - are specific to the filesystem.

- ## Rsync and hard links

  Rsync handles hard links very cleverly. Normally, if you edit a file with several links to it, you just change the contents of the file and the changes "become visible" through all its different filenames. Rsync, however, on seeing that it has to make changes to a file, first checks whether there are multiple links to the file. This information is available through the `stat()` function in both C and perl, as the `nlink` field. If there's only one link, rsync just updates the file in place. However, if there are any more links, rsync "breaks" the link by copying the file contents, unlinking it (which reduces the link count by one, and coincidentally you now know why perl and C use the oddly-named `unlink()` function to remove files), updating the file, and finally renaming the temporary file that it created[3].

- ## How rsnapshot works

  And that finally brings us to how rsnapshot works. The very first time you make a backup, rsync just copies everything to the right place, doing nothing fancy. For all subsequent backups, we first replicate the previous backup into a parallel directory structure, creating all the directories and making hard links to all the files. We then run rsync which leaves unchanged files alone and breaks the hard link for any changed files and then updates their contents. This then gives us two - or more - parallel directory trees, each of which appears to contain a full backup which makes it nice and easy to restore individual files but the amount of disk space used is only that of one full backup plus incremental backups for all the changes.

  rsnapshot also handles multiple levels of backup. So you can have, for example, seven daily backups, four weeklies, and three monthlies. When you tell rsnapshot to make a daily backup, it will first rotate the daily backups, deleting the oldest if necessary, then perform the link-and-rsync procedure described above. When you tell it to perform a weekly backup however, the weeklies are rotated and

then the oldest daily backup is renamed to be the newest weekly backup. Likewise with monthlies, the oldest weekly becomes the most recent monthly.

As you can see, most of the work is done by rsync, which perl just feeds with appropriate arguments constructed by examining a config file. Perl also handles all the directory rotation. Perl can also create all the links before calling rsync, and can delete old backups which have "rotated out". However, for speed, you can also have perl call GNU `cp`[4] and `rm` for these tasks.

rsnapshot is therefore "classic" perl. It's not really an application written in perl, it's a script using perl as a wrapper (to parse a config file and feed parameters to other commands) and as glue (calling `cp`, `rsync` and `rm`, and doing a little house-keeping in between).

Why use it

Everyone knows the value of backups and indeed of having multiple backups. Unfortunately, good backups - by which I mean frequent regular backups which are easy to restore from - are too difficult and expensive for a great many users.

Traditionally, backups have used tape, requiring expensive equipment - and without *really* expensive equipment backing up large amounts of data is not feasible. This matters, because the price of disks has plummeted recently to the point where a Terabyte costs well under £600, and with the proliferation of digital photographs and videos in the home and other digital data in business, that space gets used. Home users normally use CDs or DVDs for backup, but those are low capacity and require manual feeding of media into the machine, so while you might start off with the best of intentions, frequent regular backups often fall by the wayside. And of course with tape, CD and DVD, you can't just restore a file. You first need to find and insert the right medium. If you use incremental backups you might need to cycle through several tapes to restore one file. And CDs, DVDs and cheap tapes all fail far too often for my tastes. It's all a gigantic pain in the backside.

rsnapshot's backups live on an ordinary filesystem, usually on a cheap machine with lots of disks which is dedicated to the job. The disks are usually mirrored or use RAID5. This means that you never have to swap media, so with no manual intervention required, backups can go in cron jobs and still work even if you're still down the pub and haven't put blank media in the machine. You'll never forget to make them, or say "it can wait until the morning" and you'll get emails notifying you of any errors. Nor will you accidentally insert the wrong tape and have the backup fail - or worse, overwrite a backup. Best of all, because your backups are on a filesystem, they are instantly available. The trick with hard links means that they all look like full backups. This makes restoring files easy. You can even give your users NFS access to the backups so they can restore their own data without coming and whinging at you.

There is also support for running pre- and post-backup scripts, which are particularly useful for tasks like backing up databases.

Who uses it

Lots of people use rsnapshot for all the above reasons. Some of them may surprise you. There are several hundred people on the mailing list which all users are encouraged to join. We can conservatively assume

that as many again use it without being on the list, for a total of well over a thousand users. Most, of course, are individuals or very small companies. But we also have:

- churches
- charities
- schools
- universities
- an Amerindian tribe's casinos
- a film special-effects house
- an oil exploration company
- a floating hospital which visits African coastal countries
- a team at the Pentagon looking after GPS satellites

Why you might not want to use it

Naturally, rsnapshot, being software, has to be hateful in some way. It has three main limitations:

1. It is a filesystem backup, not a machine backup. It won't backup information like the boot record;
2. Hard links within the data being backed up are broken. You can prevent this by making rsnapshot pass the -H option to rsync, but this is not done by default because it makes things *very* slow and requires huge amounts of memory. In fact I can't use it myself because the amount of data I'm backing up makes this option require more than is possible in a 32 bit address space;
3. Strictly speaking it takes a little more space than that required for one full backup plus incrementals. A one byte change in a file - or even a change in metadata such as the owner - means that the entire file gets duplicated. This is fine for small files, or those which change seldom like applications or your /etc/passwd file. It is a problem for large files which constantly change such as your logs or your mailboxes.

and some minor ones:

1. The rsnapshot server *must* be Unix-based and the filesystem you're backing up to must support hard links - so no Windows servers, and no FAT filesystems.
2. It can only backup information that rsync knows about, so proprietary extensions like ACLs and extended attributes such as the *immutable* bit on ext2fs are not backed up.
3. Windows ownership and permissions, being ACL-based and completely incompatible with Unix filesystems, are not handled well;
4. We don't support backing up "Classic" Mac OS machines

Thankfully for most people none of these are particularly important. Restoring the boot record and OS before restoring your data is usually easy. Hard links are rarely used, being well under one in a thousand directory entries on a typical Linux machine. And on the majority of machines, logs and mboxes are but a small proportion of their data.

Design choices

The configuration file is plain text, with each line being a directive and a list of parameters. This is very

easy to parse, reading a line at a time and using split() to extract the various bits. Using this simple format instead of something more complex like a Windows-style .INI file or XML or YAML removes external dependencies. Having to use additional non-core modules would reduce the potential audience significantly, as many users don't know how to get them or aren't allowed to install them. Rsnapshot is intended to be an application that is as easy to install as something like rsync, and users should not have to care that it uses perl instead of being written in the shell or C.

Something that I consider to be unfortunate is that the config file is whitespace-sensitive - tabs and spaces matter, just like in a Makefile. It's too late to change this though.

In another attempt to make it as easy as possible to install, we are not fussy about which version of perl the user has. We support 5.6.0 at least and aim to support earlier versions. This, of course, limits what bundled modules we can use. Thankfully, finding out which modules were bundled with which versions of perl is made easy by the `Module::CoreList` module.

Aside from perl itself (of which version 5 is installed by pretty much every Unix available these days) the only other thing we require is rsync, which must be installed on both the server and on all clients. Having GNU `cp` is nice, but we have implemented our own version of `cp -al` in perl in case the GNU version is not available. Another optional extra is ssh, so that you can use rsync over the network without exposing your plain-text data. But you can use rsh or an rsync server instead.

To avoid very lengthy backups being interfered with by a subsequent backup which starts before the previous one finishes, rsnapshot uses locking so that you can never have two concurrent backups. As a backup cycle may involve expiring and deleting an old backup - indeed, *most* backups will do - and that unlinking all the files in a backup can take a very long time, there is an option to postpone deletion until after the lock has been released. This "lazy delete" is a new feature which still needs more work. It's safe to use, but not optimal.

Problems

Aside from the usual small bugs that all software suffers from - and we generally get them fixed within a matter of days - the biggest problems have come from external programs and from people not understanding what rsnapshot does and how.

A particular recent external problem came from GNU `cp`. Recent versions changed how they handle trailing slashes on directory names, causing rsnapshot to break with a rather unhelpful error message. We fixed that in CVS and told people on the mailing list, but it took some time before a new release could be bundled and published, so we had to deal with the same problem being reported over and over again. I was tempted to deprecate using `cp` but the speed advantage precludes that. And while others have suggested using `cpio -pldm` instead, there are buggy implementations of `cpio` out there too!

People often wonder why they can't do their weekly backups until they've got a full set of dailies. This is documented, but obviously not documented well enough. While the documentation is complete and accurate, new features in particular have been documented by just tacking them on the end or adding in the middle, when sometimes the users would have been better served with a re-write. Both the man page and the HOWTO document are in dire need of a rewrite.

Error reporting has not been as clear as it could be. This has improved dramatically in recent months but still needs work.

The release cycle is slow. Features can be present and thoroughly tested in the CVS version for several months before getting packaged, and many users are understandably anxious about checking the software out of CVS and just using that.

The Community and how you can help

The `rsnapshot-discuss` mailing list is reasonably active, seeing messages on most days. Traffic is mostly people reporting problems (usually, as noted above, because they haven't understood the documentation) and being helped out by other list members. Discussion of new features, bug reports, patches, and release announcements make up the bulk of the remainder. I am very grateful that other users take the time to answer most questions, even the ones that come up over and over again.

I'm also lucky that the people submitting bug reports often take the time to dig into the code and try to figure out where the bug is and sometimes even include a patch. Even when they don't do that, the bug reports are usually accurate and concise.

A handful of people other than myself have commit access to CVS, and I'm liberal about handing out that permission - if someone has sent a good patch to the list, then I'll give them CVS commit rights if they want. One of the committers concentrates particularly on database support scripts.

What we really lack at the moment is someone with the time and inclination to go through the documentation and make it more useful to new users. But any other contributions of expertise, patches, or code review would be most welcome.

Footnotes

1. http://samba.anu.edu.au/rsync/
2. Easy Automated Snapshot-Style Backups with Linux and Rsync; Rubel, Mike; http://www.mikerubel.org/computers/rsync_snapshots/
3. Actually that's not true, it *always* makes a new copy and then moves it into place, as that means files are never in a half-changed state. But the effect is what I described. Rsync lets you override this with the `--inplace` option, but obviously you shouldn't do that in rsnapshot!
4. This must be GNU `cp`, which supports the non-standard `-l` argument to create a link to a file instead of creating a new copy of a file. In tests this has been up to 50% faster than walking the directory tree and calling `link()` in perl.