BACKING UP LINUX - II

1 Software recommendations, disrecommendations and examples

This chapter describes some details and examples of a number or programs. Please note that there are far more backup programs in existence than the ones I mention here. The reason I mention these, is because these are the ones I have a lot experience with, and it shows how to apply or consider the concepts described above, in practice.

Sometimes as specifically refer to the GNU version of an application, the standard version on Linux installations, which can be fundamentally different than the classic version, so keep that in mind.

1.1. Dar

First a word of caution. It's highly recommended that you use version 2.3.3 (most recent stable release at the time of writing) or newer because it contains a major bugfix. Read the announcement on Dar's news list for more info.

Dar is very well thought through and has solutions for classic pitfalls. For example, it comes with a statically compiled binary which you can copy on (the first disk of) your backup. It supports automatic archive slicing, and has an option to set the size of the first slice separately, so you have some space on the first disk left for making a boot CD, for example. It also lets you run a command between each slice, so you can burn it on CD, or calculate parity information on it, etc. And, very importantly, its default options are well chosen. Well, that is, except for the preservation of atimes (see atime-preserveration above).

I use the following command to backup my system on an external USB drive about once a week, using dar 2.2.6 (most site specific options removed, and abstracted a bit):

```
           dar --execute "par2 c -r5 \"%p/%b.%n.par2\" \"%p/%b.%n.%e\"" --
alter=atime --empty-dir \
               --fs-root / --noconf --create ARCHIVE_NAME --slice 620M --first-
slice 600M -z6 \
               -an -Z "*.ogg" -Z "*.avi" -Z "*.mp?" -Z "*.pk3" -Z "*.flac" -Z
"*.zip" -Z "*.tgz" \
               -Z "*.gz" -Z "*.gzip" -Z "*.bz2" -Z "*.bzip2" -Z "*.mov" -Z
"*.rar" -Z "*.jar" \
               --prune lost+found --prune usr/portage/ --prune var/tmp/portage -
-prune media \
               --prune proc --prune mnt --prune sys
```

With the *--execute* statement, I calculate parity information with par2. The mystery-strings passed to par2 translate into the par2 file(s) to be generated, and the archive slice name. *--alter=atime* Is mentioned above. The*--empty-dir* option stores every excluded dir as an empty dir in the archive. The *-an* and subsequent *-Z* options specifies what to exclude for compression, with case insensitive masks. The compression level is specified with *-z6*. *--prune* Is used to exclude paths. The rest should be clear.

I also used to run a daily backup with dar on */home*, without compression. This was about 6 GB and takes about 10 minutes. Very feasible I would say. However, when my home dir grew in size, I switched over to Rsync and later to rdiff-backup, and accepted the change detection flaw.

Restoring a dar archive should be safe with the defaults (a very important aspect in my opinion), but read the man page to be sure.

1.2. GNU Tar

GNU Tar supports everything needed to make a reliable backup (although I must say I don't know how well, if at all, it supports extended attributes). One has to be careful though, to supply the *--numeric-owner* option. This could also include *--same-owner*, but a quick test and glance at the manual shows the *--preserve-permissions*option (which is enabled by default for user root) implies this. Should you have forgotten the *--numeric-owner*option for the backup command, this can also be given at restore time. Giving it at backup time, *should* negate the necessity of giving it at restore time because tar won't store the textual names in the archive when you give this option.

Tar also has no decent splitting ability. I've seen people recommending to use split, but that is not very convenient. With split, you first have to create the archive somewhere where it fits, and then split them up to be stored somewhere else. At restore time, this is even more annoying, because you first have to concatenate the segments with cat before you can extract them.

An additional problem with Tar was reported to me by a reader. With version 1.15.1 he got corrupted sparse files which were bigger than 4 GB. Current releases (1.20+) of Tar don't seem to have this problem anymore, but you might want to check for yourself.

I recommend not using tar unless you can live with the limitations are and careful to supply the correct options.

1.3. Rdiff-backup

You have to consider its method of change detection to determine if rdiff-backup will be reliable enough for you. If you deal with disk images a lot, read the

example above. I once discussed alternative change detection methods with the author, but because of lack of time on his part, the discussion was never really concluded. It can very well be that in the future it will include a reliable change detection system, like hashes or ctimes. Note that rdiff-backup does store the hash in its meta data since version 1.1.1 (2005/11/05), but it still doesn't use it for detecting changes in files (at time of version 1.2.1, 2008/08/24).

Also, when restoring a full system backup from a different OS installation, like a LiveCD, with rdiff-backup, be sure to use the --preserve-numerical-ids option, otherwise you will end up with files with wrong owners. This is very easy to forget (I've done it myself).

In the mean time, if you are confident enough you won't suffer from the mtime problem described above, in your*/home* for example, it can safely be used. You may also find my concerns exaggerated and use it for your entire file system after all. I decided these concerns weren't big enough to stop me from using it. Rdiff-backup is a very reliable and one of the best incremental backup programs, in my opinion.

1.4. Rsync

My main problem with rsync is that it stores its meta data as new meta data in the target file system. Not only does this restrict the use of target file system, but it is also somewhat flaky, as described above. Also, rsync is one of the tools which uses the mtime+size for detecting changes in files. And because its options for reliable checks for file changes (with --ignore-times or especially --checksum) are too slow, it can make dar without compression a better choice. That is, only when backing up locally on a fast medium, of course.

An important thing to note here, is that because rsync doesn't store meta data files, it compares the mtime+size with the target. This means that only when you use the --times option, to preserve the mtime of the files, does this change detection fail in the scenario described above.

Rsync has a special switch, *--archive*, specifically meant for preserving all meta data. Using this flag is not enough, however. For instance, it doesn't store hard link information by default, because that would be too slow. Also, it doesn't store hard links, extended attributes and access control lists. So, you need to supply *--hard-links* as well, and *--acls* and *--xattrs* should you use them (rsync supports extended attributes since version 3, I think). The options *--sparse* and *--numeric-ids* are also recommended, as outlined above. Additionally, I would supply *--delete --delete-excluded --delete-after* as well, to not get stale files in my backup. The *--delete-after* is necessary because otherwise, should the backup fail half-way, files that have been renamed since the last backup (meaning, old file is deleted, new file is created), are deleted before the new one is transferred. It's best to first transfer the new file, then delete the old one.

It is also important to construct a proper restore command. Because the destination should be an exact mirror of the source, using the same options as you use for making the backup, is probably enough.

Some time ago, Rsync 3 was released. It contains interesting new features, such as support for access control lists and extended attributes. It may have the issue with the change detection, but especially since version 3, you might want to consider using it (perhaps only for a part of your backup plan), because it's still very efficient and thorough.

1.5. GNU cp

I always thought that *cp -a* would preserve everthing you need, but it appears it doesn't copy access control list information, and probably extended attributes in general. I don't use access control lists, so I can't do any tests with them, so you want to test this yourself.

1.6. Clonezilla

Clonezilla is an elaborate live-CD for making images of several types of file systems, including NTFS. Besides some interface quirks, it's a beautifully engineered piece of software. It does all the extra things one expects, like using dd to backup the MBR and the space between the MBR and the first partition. It even calls *sync* when it's done. It's almost as if they read this article :). And, perhaps the most important: it allows you to remove the CD when halting or rebooting!

1.7. Partition cloners in general

Partition cloners, like g4u, partimage, clonezilla (or dd...) can be very convenient, but they (mostly) have one major annoyance: they (often) require that the backup is restored on an identical disk and/or partition layout. If your disk explodes and you need to get a new one, this can be difficult.

However, this is not always the case. I recently just dd'ed an entire disk over to another one (that was 1 GB bigger), with `dd if=/dev/sda of=/dev/sdb`. The new one now has unpartitioned space, but it still works fine. The Windows XP installation that it contained, still boots on the new drive, even though Windows (XP) is very picky about that. In any case, restoring to a smaller partition can be a problem. Sometimes you can improvise, but this is something best avoided. If you are going to use this method of backing up, it's a good idea to make sure replacement disks are always big enough. You can do this by assuming disks will always get

bigger in the future or by keeping your partitions small (but not too small as not to inhibit fragmentation prevention).

The less intelligent cloners, like g4u or dd, take an enormous amount of time, because they copy every file system block, including the ones that are not used. And, when these unused blocks are not zeroed out first, the resulting image is very large.

Another aspect most people will find annoying, is that you have to take your machine down to make the image. To my knowledge, there are no partition imagers which can do this on a live system. In any case, you certainly shouldn't use dd on a live partition...

---

2. Automation

When automating backups, there is one important thing you have to keep in mind: sync the disk buffers as often as possible, and preferably wait a second or two after the sync, for the drive to write its own cache to the disk itself. Some disks lie about having written their cache to disk, so it's not always safe to assume that all the volatile cache data has been written after the sync command returns, hence the wait.

To illustrate why this is important, consider a tar backup routine which first makes the backup and then removes the old one. If the cache isn't synced and the power fails during removing of the old backup, you may end up with both the new and the old backup corrupted. It has been suggested to me that this is pointless, because cache data is serialized, so any delete after the initial backup command first writes the volatile cache. This is not the case, especially if you have a disk

with NCQ/TCQ, which all modern disks have. The whole point of the write cache is to be able to write out of order.

My suggestion is, to run the backup like this:

```
sync ; sleep 2
mount target
[BACKUP-COMMAND]
sync ; sleep 2
[CLEAN-COMMAND]
sync ; sleep 2
umount
```

The sync at the end is also important, so that when the unmount fails, the data is still safe when the drive is unplugged or whatever.

---

3. File system replication

If you want to be able to take snapshot of your entire file system, you may want to consider using logical volumes, with LVMs. With logical volumes, you can take atomic snapshots of the entire volume. Because the best backup is done at or below the file system, in term of retaining all (meta) data, this provides a very robust way of making backups.

However, I have never used LVMs, so I can't really say anything useful about them. You will have to read up on the specifics to find out if they actually do what you want.

Should you make use of snapshotting, precautions still have to be taken for applications that can change their data files during the backup. You should still

make dumps of your databases, LDAP trees, etc, because you don't want to capture their files in the midst of a transaction.

---

4. Choice of file system

Although it is somewhat out of the scope of this article, I would like to say a few words about the choice of file system. People often choose Reiser FS over Ext3, because it's new or just different. The default file system on most Linux systems is Ext3. My recommendation is to stay with that, unless you have a specific reason to use something else. Reiser FS, for example, does logical journaling. According to this, this can be dangerous in the event of a power failure. Also, Hans Reiser himself has said (or so it's stated) that Reiser FS is optimized for speed, not correctness.

Even the new Ext4 file system has potential problems, because of delayed allocation. Ext3 has it as an option (data=writeback), but for Ext4, it's the default. Delayed allocation means, in the case of Ext4, that the data is written to disk in the order of once a minute, whereas the meta data is written much more often, in the order of every few seconds. Linus Torvalds says: "It literally does everything the wrong way around -- writing data later than the metadata that points to it. Whoever came up with that solution was a moron. No ifs, buts, or maybes about it. At least with ext3 it's not the default mode."

Anyway, my point is to investigate any file system you may want to use. And, Ext3 may be ordinary, but it is not that bad. The references chapter has some links to more information about the subject.

Source : http://www.halfgaar.net/backing-up-unix