Backup Options For FreeBSD

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
	2012-12-20		WB

Contents

Last updated 2012-12-20

Available in HTML or PDF. Links to all my articles here. Created with AsciiDoc.

1 Introduction

There are many ways of backing up a FreeBSD system. Which method to choose depends on your needs. Here we'll describe the most common methods, and their benefits and disadvantages. The FreeBSD Handbook Backup Basics chapter is another source you should examine.

Be very careful about backups. Test the data from them. The more copies, the better. Consider the information here advisory and untested. No guarantee is expressed or implied that any of this works or will protect your data or will elicit anything more than loudly apologetic noises if data loss occurs.

2 Archives

Beware of backup methods that only keep one copy of data; there's no way to retrace your steps. Deleted a file last month? It's gone, because we only have a backup from last night. The backup methods below save to files, and it's easy to put those files into a new yyyy-mm-dd directory each time. Keep backups as far back as possible.

3 *dump(8)*/*restore(8)*

dump(8) and *restore(8)* have been the standard FreeBSD backup utilities since vacuum tubes roamed the earth. They work at the filesystem level, and so only back up used space. `restore` can provide access to individual files in a backup, useful when your system is fine but you just need that one file out of a backup.

`dump` is not particularly fast, but because it only backs up actual data (filesystem blocks in use), can be faster than whole-disk backup methods.

Each filesystem must be dumped individually; this is sometimes expressed as "`dump` does not cross filesystems". On a normal FreeBSD system, a full backup includes dumps of */*, */var*, and */usr*. */tmp* is a separate filesystem, too, but you're not supposed to be storing valuable data there.

/ and */var* are usually small and trivial to dump, with */usr* being much larger. If you're dumping to files on a FAT filesystem, the 2G/4G file size limit can be an unwelcome surprise.

`dump` can take either mountpoints like */var*, or the device node of an unmounted filesystem, like */dev/ada0s1d*. If a filesystem is mounted, the *-L* snapshot option should be used.

3.1 Useful `dump` Options

One of the nice usability features of FreeBSD is that ctrl-T sends a SIGINFO to the controlling process. With `dump`, this means you can press ctrl-T and get a "percent completed" report. That report will also be printed automatically every five minutes, if you're the patient sort.

Giving `dump` a larger cache with the *-C* option can help speed things up. Be a little cautious: since `dump` forks multiple processes, the cache size will be used multiple times. If you overcommit memory and the computer starts to swap, the dump will change from "annoyingly slow" to "glacially agonizing". The *dump(8)* man page recommends 8 to 32 megabytes. It also recommends always using *-C* when the *-L* snapshot option is used.

-b64 is also used to increase buffer size and improve performance.

Directories to be skipped in a dump can be marked with the *nodump* flag:

```
# chflags nodump /usr/ports
```

Because the Ports Collection is easy to recreate and often contains large distfiles, it's a prime candidate to skip in backups. (On computers with poor net access where those distfiles are hard to download, you might want to back it up; as always, it depends on your situation.)

Other large directories with easy-to-recreate or unnecessary data could include */usr/src* and various types of cache, like web browser cache or the files created by *devel/ccache*.

3.2 *restore(8)*

`restore` has many useful options, including the ability to interactively choose individual files for restoration. Here, we'll only show restoring a whole filesystem, but please see the *restore(8)* man page.

Caution

`restore` puts restored files in the current directory. Remember to `cd` to the right directory before running `restore`.

3.3 Simple `dump`

Back up a single system to an external hard drive. The external hard drive is UFS-formatted, so file size limits are not a problem.

```
# mount /dev/da0s1 /mnt
# dump -C16 -b64 -0uanL -h0 -f /mnt/root.dump /
# dump -C16 -b64 -0uanL -h0 -f /mnt/var.dump  /var
# dump -C16 -b64 -0uanL -h0 -f /mnt/usr.dump  /usr
# umount /mnt
```

The external drive is mounted at */mnt*, then the */*, */var*, and */usr* filesystems are dumped to files on it.

Restoring this backup is also pretty simple. This example restores the */var* filesystem into a temporary directory. This is the type of restore used when you're looking for some files that were accidentally deleted. If you were restoring a whole system, you'd restore the data over the filesystem where it originated.

```
# mount /dev/da0s1 /mnt
# mkdir /tmp/oldvar
# cd /tmp/oldvar
# restore -ruf /mnt/var.dump
# umount /mnt
```

See the *restore(8)* man page for the *-i* option, which provides a shell-like interactive session for selecting files and directories to restore.

3.4 `dump` With Compression

```
# mount /dev/da0s1 /mnt
# dump -C16 -b64 -0uanL -h0 -f - /    | gzip -2 > /mnt/root.dump.gz
# dump -C16 -b64 -0uanL -h0 -f - /var | gzip -2 > /mnt/var.dump.gz
# dump -C16 -b64 -0uanL -h0 -f - /usr | gzip -2 > /mnt/usr.dump.gz
# umount /mnt
```

Like the previous example, but `dump` output is piped through `gzip`, creating compressed dump files.

`gzcat` is used to decompress these files for restoration. This example restores to a temporary directory so the backed-up directories and files are available but don't overwrite the current files.

```
# mount /dev/da0s1 /mnt
# mkdir /tmp/usr
# gzcat /mnt/usr.dump.gz  | (cd /tmp/usr && restore -ruf -)
# umount /mnt
```

3.5 dump Via SSH

ssh allows dump to send files to another system on the network, which is very handy, at least if you have more than one system.

```
# dump -C16 -b64 -0uanL -h0 -f - /    | gzip -2 | ssh -c blowfish user@otherhost dd of=root ↩
    .dump.gz
# dump -C16 -b64 -0uanL -h0 -f - /var | gzip -2 | ssh -c blowfish user@otherhost dd of=var. ↩
    dump.gz
# dump -C16 -b64 -0uanL -h0 -f - /usr | gzip -2 | ssh -c blowfish user@otherhost dd of=usr. ↩
    dump.gz
```

otherhost is the system receiving the backup files, and *user* is the username on that other system. As before, gzip compresses the files, but then ssh logs in to the other system and copies the input into a file in the user's home directory with dd.

Backup speeds may be slower than to a directly-connected disk. dump and ssh prompts are mixed together, which can be confusing when ssh is waiting for a password. But the convenience of this method sometimes makes a FreeBSD notebook or netbook a better "backup device" than a simple external hard disk.

Restoring these files is a reversal of the dump command line:

```
# mkdir /tmp/root /tmp/var /tmp/usr
# ssh -c blowfish usr@otherhost gzcat root.dump.gz | (cd /tmp/root && restore -ruf -)
# ssh -c blowfish usr@otherhost gzcat var.dump.gz  | (cd /tmp/var  && restore -ruf -)
# ssh -c blowfish usr@otherhost gzcat usr.dump.gz  | (cd /tmp/usr  && restore -ruf -)
```

Depending on the speed of the computers and the network, it may be faster to run the dump or restore on one computer and the compression on the other. For example, a restore that sends the compressed dump file over the network and decompresses it on the destination computer:

```
# ssh -c blowfish usr@otherhost dd bs=64k if=usr.dump.gz  | (cd /tmp/usr && gzcat | restore ↩
     -ruf -)
```

There are many possible variations. Networks and computers vary so widely that only testing will show what is effective on a particular system.

3.6 Copying Filesystems

Filesystems can be copied directly from one disk to another by piping the output of dump right into restore. Source and target filesystems don't have to be the same size, the target just needs to be large enough to hold all the data in the source.

This example copies a running FreeBSD system onto a new drive. Slices and partitions have already been set up on the new disk, *da0*. The target is the first slice, with a traditional FreeBSD partition layout: *a* is */*, *d* is */var*, *e* is */tmp*, and *f* is */usr*. See Disk Setup On FreeBSD for background.

/var and */* are copied last. They are smaller and may have changed during the copy of the much-larger */usr* filesystem.

```
# mount /dev/da0s1f /mnt
# dump -C16 -b64 -0uanL -h0 -f - /usr | (cd /mnt && restore -ruf -)
# umount /mnt
# mount /dev/da0s1e /mnt
# dump -C16 -b64 -0uanL -h0 -f - /tmp | (cd /mnt && restore -ruf -)
# umount /mnt
# mount /dev/da0s1d /mnt
# dump -C16 -b64 -0uanL -h0 -f - /var | (cd /mnt && restore -ruf -)
# umount /mnt
# mount /dev/da0s1a /mnt
# dump -C16 -b64 -0uanL -h0 -f - /    | (cd /mnt && restore -ruf -)
# umount /mnt
```

3.7 *restore(8)* Over A Live System

It is possible to restore over a live system. For example, if a hard drive dies, do a minimal FreeBSD install on a new hard drive, then boot that system and `restore` the */usr*, */var*, and then */* filesystems. I suggest that order because continuing to run after overwriting important data in */* is probably something to avoid. After the restore and a reboot, doing a full *buildworld/kernel/installworld* cycle is a good idea, to make sure the whole system is consistent.

The New Huge Disk section of the FreeBSD FAQ shows a different technique for a similar operation.

Tip
/etc/fstab can be a tricky file. Restoring a backup of the */* filesystem to a different computer writes this file, but chances are good that the device names or numbers it uses to refer to filesystems will be different on the new computer. Device or filesystem labels are a convenient way to avoid the problem.

4 Clonezilla

Clonezilla is an open source live CD/USB image that has some of the disk copying functionality of Ghost. It's not as easy to use; you need to know a few Linuxisms like that external hard drive you just connected is */dev/sda* or *sdb*, or maybe *hda*. But once you get past that, it has lots of useful features. Several open-source backup programs are used to make sure that only occupied space is copied. Data is compressed and split into 2G files so it can use FAT filesystems for backup. It can use external drives, or SSH, Samba, or NFS network drives. It's menu-driven, so it's not terribly difficult to use. And the latest versions even support FreeBSD's UFS filesystems.

Individual files aren't accessible in the backups, and there's no way to skip directories that don't need to be backed up. But for whole-disk, mostly-automated backup of FreeBSD systems, you could do worse.

5 `dd`

`dd` is not a good way to back up a system. It has many disadvantages, including copying every block on the drive or slice, whether used or not.

The advantages are that `dd` is very easy to use. Just copy one drive to another—what could go wrong?

Well. . . the target drive must be at least as large as the source drive. It can be larger, although any extra space will be wasted.

Be very careful about source and destination drives. Get them mixed up on the command line, and the source drive will be overwritten. Allow me to repeat that: getting source and target mixed up will destroy your data.

Some notable utilities are based on using `dd` like this. g4u (Ghost For Unix) is one of the better-known versions.

5.1 Simple `dd` Usage

Copy drive *ad0* to *ad4*. *ad0* is your source drive, right? And you're positive about that, right? The drive numbers didn't change after adding the second drive? Okay. . .

```
# dd if=/dev/ad0 of=/dev/ad4 bs=64k
```

Setting the *bs* (block size) parameter to 64k will buffer the reads and writes and speed up the copy. Why not an even larger buffer? Because a larger buffer won't speed up a typical drive. If you have a RAID setup or something faster than typical SATA or IDE, a larger buffer might be useful.

5.2 `dd` With Compression

A lot of space on the average disk is empty, but the blocks still have old data in them. That random old data doesn't compress well if you `dd` that drive into an image file. You can improve the situation by filling all unused space with zeros. The brute-force method—and `dd` is nothing if not brute force—is to just build a file full of zeros until you completely fill the disk. Then delete the file.

```
# dd if=/dev/zero of=zerofile bs=1M
  ...(wait for an out-of-space error)
# rm zerofile
```

Unused space on the drive is now mostly compressible zeros.

```
# mount /dev/da0s1 /mnt
# dd if=/dev/ad0 bs=1M | gzip -2 > /mnt/diskimg.gz
# umount /mnt
```

While the image file is smaller, restoring it will still require a drive at least as large as the original.

6 Other Programs

There are advocates of all sorts of programs for backup. *cpio(1)*, *pax(1)*, and *tar(1)* are mentioned frequently in this context. For some things, they work fine. But only `dump` backs up at the filesystem block level and can be trusted for any of the weird situations a UFS filesystem may contain.

There are other situations, of course. For making an archive of a directory tree, *tar(1)* with a compression option like *-z* or *-j* is a widespread standard.

For copying files or directories, *net/rsync* is very useful. It only copies the differences between source and target, and supports compression, so updates are very quick. Built-in support for *ssh(1)* means those directories can be on separate computers on the network. If you wanted to make a backup of selected directories of data files, `rsync` works extremely well for that, and it's versatile enough to be used in lots of ways.

7 Conclusion

There are many ways to go about keeping data that was expensive and difficult to create, some that go far beyond the ones listed above. There's Bacula, and Amanda, and all sorts of commercial and homebuilt schemes.

Whatever you pick, remember that backups are the kind of thing you only need when you don't have them.