Using Dump & Restore for Backups

Dump and Restore on OpenBSD and Ubuntu Linux Distros

What is Dump and Restore?

Dump is a software for doing backups of the files on your Linux or OpenBSD computer.

Restore is the software that restores those backups. Restore can restore the entire backup or a portion of it in interactive mode.

In OpenBSD Dump and Restore are part of the base system. For Ubuntu Linux, Dump has to be installed with apt-get:

```
# sudo apt-get install dump
```

the associated Restore program is part of the Dump package and will be installed at the same time as Dump.

The Benefits of Dump and Restore

The dump command builds a list of files that have been modified since the a previous dump, then packs those files into a single file to archive to an external device. Dump has several advantages over other utilities:

- Files of any type (even devices) can be backed up and restored.
- Permissions, ownerships and modification times are preserved.
- Files with holes are handled correctly.
- Backups can be performed incrementally[1].

1. From UNIX System Administration Handbook, by Evi Nemeth, 3rd Ed., pg. 176.

Backing up files with Dump

These instructions presume you are using an external HDD.

To dump files on OpenBSD, use the following command:

dump -0 -f /mnt/feb.dump jon/Howard.rtf jon/howard.*

this command copies all files named howard.* and the sole rtf, Howard.rtf
to the external hard drive.

Another dump:

Unmount /home so that no activity is going on. /dev/rwd0g is the partition that /home resides on.

# umount /home

Then do: # dump -2au -f /mnt/bsd/OpenBSD_lvl2.dump /dev/rwd0g

This copies the latest additions to the home directory (/dev/wd0g) to the file OpenBSD_lvl2.dump, a level two dump. The switches -2 gives the dump level and -a means bypass tape length calculations. The switch -u means update /etc/dumpdates

Backing up files with Dump

The most important options for dump are the following:

-0, -1, -2, …. -9

These options indicate the level of the dump this command will perform. Given any level n, dump will search dumpdates for an entry reporting the last time this filesystem was dumped at level n-1 or lower. dump then backs up all files that have been changed since this date. If n is zero, dump will back up the entire file system. If there is no record of a backup for this file system for level n—1 or lower, dump will also back up the entire filesystem. If no level option is specified, it defaults to -9. This option does not require any argument.

-u

If dump finishes successfully, this option updates its history file dumpdates (in OpenBSD: /etc/dumpdates). It does not require an argument,

Backing up files with Dump

The most important options for dump are the following:

-f

This option states that you want to send the dump to something other than the default tape drive (i.e., to a file or to another device). If you use this option, it must have an argument, and this argument must precede the filesystem being dumped. A value of “-” (a single hyphen) for its argument indicates standard output.[2]

Example:

```
# dump -0 -a -u -f /mnt/OpenBSD_lvl0.dump /dev/rwd0g
```

- This is a level zero dump
- It is dumping to an external hard drive (-f switch)
- -u means update /etc/dumpdates
- -a “auto size”. Bypass all tape length considerations, and enforce writing until an end-of-media indication is returned

[2] Essential System Administration, Aeleen Frisch, 3rd Edition, page 733

What is the problem when dumping live filesystems?

From Source Forge “Is Dump Deprecated”

The problem is that the filesystem may be changing while you are dumping it. You have this problem with all backup utilities, but with dump it is more serious. When you are using tar, for example, a file could be changed at the time it is read by tar; in that case, that particular file would be corrupted in the resulting tar file. But whereas for tar this is a problem only if it so happens that the file is changed the instant it is read, dump could backup corrupted versions of files if they changed some time before dump attempts to read them. Let's see why.

The kernel caches write operations to the disk. You can see this for yourself if you make some experiments with a floppy. Insert a floppy in the drive, mount it, and copy a file to the floppy; the operation, especially with recent 2.4 kernels, will appear to finish instantly. You can then do something like ls /mnt/floppy and see that your file is on the disk. But your file is not really on the disk; the drive's light hasn't been on at all. If you looked at the disk through /dev/fd0, you wouldn't find your file there.

You can force the file to be actually written to the diskette by unmounting the diskette, or with the sync command; if you don't, the kernel will write the file on the disk when it sees it fit to do so. What's more, the kernel might actually write half the file on the disk, and I guess that this will be usual with hard disks; when there are lots of pending write operations at approximately the same physical area of the disk, the kernel will probably choose to flush them, but it will probably choose not to flush other pending operations at distant areas, so as to minimize head movement. Thus, when dump reads the filesystem through the block device, it will get corrupted versions of some files if there are pending write operations; even worse, the metadata (filesystem structure) could be corrupt, in which case (a part of) the filesystem could become entirely unreadable.

What is the problem when dumping live filesystems?

From Source Forge “Is Dump Deprecated” **Can I use dump, then?**

Dump is a really popular backup solution among Unix system administrators worldwide, and it is not because those administrators are ignorant of the problems.

First, you can safely use dump on unmounted and read-only filesystems. You can also safely use dump on idle filesystems if you sync before dumping (but can you be sure they are idle? a solution is to remount them read-only before dumping).

You can also use dump on non-idle filesystems, but with caution. You must take care to dump when the filesystem is not heavily loaded; for example, I dump during the night, when only logfiles and mailboxes are modified, and not heavily. If your filesystem is always on heavy load, maybe you shouldn't use dump. In addition, you should verify your backups.

Doing a dump on an idle but live file system is sometimes neccessary as the default install on manyLinux distributions hang everything off “/” Like so:

```
$ df -h
udev             7.7G     0  7.7G   0% /dev
tmpfs            1.6G  1.8M  1.6G   1% /run
/dev/nvme0n1p5  179G   68G  102G  41% /
tmpfs            7.7G     0  7.7G   0% /dev/shm
```

The whole file system on my Mint Linux hangs on /dev/nvme0n1p5. Unmounting “/” will shutdown your computer

What is the problem when dumping live filesystems?

From Source Forge "Is Dump Deprecated" **If dump has these problems why not use something else?**

1. The fact that dump reads the block device directly gives it several advantages. First, you can dump unmounted file systems. It has been reported that this is particularly useful in cases of filesystem error which renders it unmountable; in those cases, it is useful to dump the filesystem (to the extent possible) before attempting to fsck it, in case fsck causes further data loss.
2. Dump never changes the filesystem while dumping it. The problem with tar and cpio is that they change a normally mounted read-write filesystem while reading it. The filesystem keeps three times for each file: the last modification time (mtime), the last access time (atime), and the last i-node modification time (ctime). When you read a file through a normal system call, its atime is set to the time of the access. You could then issue another system call to revert atime to its original value, as GNU tar does when given the --atime-preserve option, but in that case ctime changes to indicate an i-node modification. There is no system call to change ctime.
3. Dump's third advantage is that it works faster, because it bypasses the kernel's filesystem interface. I don't have any experience on this, but I suspect that now that the machines are faster and the filesystem caches are much larger, this advantage is less important than what it used to be.

Only level 0 dumps can be performed on individual directories. An incremental dump could be done on /dev/nvme0n1p5 but not on /home

```
$ sudo dump -0a -f LinuxMintHomeLvl0.dump /home
  DUMP: Date of this level 0 dump: Fri Aug  5 13:13:16 2022
  DUMP: Dumping /dev/nvme0n1p5 (/ (dir home)) to LinuxMintHomeLvl0.dump
  DUMP: Label: none
  DUMP: Writing 10 Kilobyte records
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 42326263 blocks.
  DUMP: Volume 1 started with block 1 at: Fri Aug  5 13:13:16 2022
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: 28.89% done at 40755 kB/s, finished in 0:12
  DUMP: 57.97% done at 40896 kB/s, finished in 0:07
  DUMP: 87.88% done at 41328 kB/s, finished in 0:02
  DUMP: Closing LinuxMintHomeLvl0.dump
  DUMP: Volume 1 completed at: Fri Aug  5 13:30:23 2022
  DUMP: Volume 1 42313570 blocks (41321.85MB)
  DUMP: Volume 1 took 0:17:07
  DUMP: Volume 1 transfer rate: 41201 kB/s
  DUMP: 42313570 blocks (41321.85MB) on 1 volume(s)
  DUMP: finished in 1027 seconds, throughput 41201 kBytes/sec
  DUMP: Date of this level 0 dump: Fri Aug  5 13:13:16 2022
  DUMP: Date this dump completed:  Fri Aug  5 13:30:23 2022
  DUMP: Average transfer rate: 41201 kB/s
  DUMP: DUMP IS DONE
```

Only level 0 dumps can be performed on individual directories. An incremental dump could be done on /dev/nvme0n1p5 but not on /home

From the OpenBSD man page (man -s 8 dump)

For individual files or directories only level 0 dumps are allowed;

```
 # dump -0 -f OpenBSDFiles_Lvl0.dump /home/files-to-dump
```

files-to-dump is either a mount point of a filesystem or a list of files and directories on a single filesystem to be backed up as a subset of the filesystem. In the former case, either the path to a mounted filesystem, **the device of an unmounted filesystem** or the disklabel(8) UID can be used. ***In the latter case, certain restrictions are placed on the backup: -u is ignored, the only dump level that is supported is -0, and all of the files must reside on the same filesystem.***

A ksh script for doing backups with Dump

```
#!/bin/ksh
# Script to backup your computer with dump (man -s 8 dump) on OpenBSD
trap 'print "You must connect the external hard drive"' ERR
# Mount the external hardrive
mount -t ffs /dev/sd1i /mnt
# Unmount the home partition, /dev/rwd0g,so it is not being written to.
umount /home

# Present choices for dump levels
PS3="Enter dump level: "

select Level in 0 1 2 3 4 5 6 7 8 9 Quit
do
case $Level in
   [0-9]) dump -${Level}au -f /mnt/OpenBSDHome-lvl${Level}.dump /dev/rwd0g;;
   Quit)
   umount /mnt
   mount /home
   exit;;
esac
done
```

Restoring files with Restore in Interactive mode (-i)

If your backup source was home, #dump -0 -f /mnt/OpenBSD.dump /home, then cd /home/

1. As root run # restore -i -f /mnt/OpenBSDHomeLvl0.dump
2. The root prompt will change from # to restore>
3. At the restore> do ls. This will show someaccount as the directory
4. Do restore> cd someaccount.ls will show the files
5. Now do restore> add file1.txt do the same for file2.txt and file3.txt say.
6. Do restore> ls and all the *.txt files have a leading asterix like so: *file1.txt,*file2.txt, *file3.txt etc. The asterix shows that the files have been marked for extraction to /home/someaccount
7. Now do restore> extract.The restore will ask "Specify next volume". In this case there is only one volume, so enter 1. If there are multiple volumes then you want to begin with the last one first.
8. Restore will then ask set owner/mode for '.' ? [y/n] and you want to use 'y'.
9. Restore will then extract the files into /home/someaccount
10. Finally do restore> quit.

Backup Strategies

Both Aeleen Frisch and Evi Nemeth have good advice on when and at what levels to do backups. Here I will quote the OpenBSD man page (man -s 8 dump).

- Always start with a level 0 backup, for example: # /sbin/dump -0u -f /dev/nrst1 /usr/src. This should be done at set intervals, say once a month or once every two months, and on a set of fresh tapes that is saved forever.
- After the level 0 dump, backups of active file systems are taken on each day in a cycle of a week. Once a week, a level 1 dump is taken. The other days of the week a higher level dump is done. The following cycle needs at most three tapes to restore to a given point in time, but the dumps at the end of the weekly cycle will require more time and space: 1 2 2 2 2 2 2. This sequence requires at most eight tapes to restore, but the size of the individual dumps will be smaller: 1 2 3 4 5 6 7. This sequence seeks a compromise between backup and restore effort: 1 2 2 3 3 4 4. The weekly level 1 dumps should be done on a set of tapes that is used cyclically. For the daily dumps a tape per day of the week can be used.
- After several months or so, the daily and weekly tapes should get rotated out of the dump cycle and fresh tapes brought in.