Using JFS2 snapshots on AIX 6.1

Learn how to use JFS2 snapshots to aid in backup and restore tasks on AIX 6.1.

Chris Gibson June 01, 2010

This article describes how to use JFS2 snapshots to backup and recover files and file systems.

Introduction

I often need to make a change to an AIX system that will require a backup of certain files or file systems. Making a copy or backup of files is considered good practice. For example, if I need to edit a series of configuration files, I would backup the files before I make any changes. Typically, I would use a familiar backup method such as mksysb, savevg, tar, backup, or even the cp command. For larger backups, I may use a licensed product, such as IBM Tivoli Storage Manager. These all work fine in nearly all cases. A potential drawback with these methods is that they can take a long time to execute and even longer to restore from, especially if recovering from tape media.

Fortunately, a feature called **JFS2 snapshots** is able to assist in making quick and painless backups. It does not negate the need for performing regular backups to tape (or offsite media), but it is suitable under certain circumstances. Restores are also faster and easier when using a snapshot. I have found that using this new snapshot technology has allowed me to perform some tasks a lot more efficiently. For example, using snapshots, I can speed up the backup and (if necessary) the recovery times when I need to modify a number of configuration files or perform an upgrade of an application.

In the following examples, I demonstrate how I used JFS2 snapshots to backup and recover a file system during an upgrade activity.

Overview of JFS2 snapshots

The snapshot functionality was provided so that administrators could create consistent, integrated snapshots of an online (or offline) JFS2 file system. A point-in-time copy of the source file system is created. The snapshot operation executes quickly and requires very little disk space. Multiple snapshots can be taken, which can be very handy if you need to perform multiple changes to a system with corresponding backups at each stage.

The snapshot retains the same permissions as the original file system. Once the image has been created, you can use it to restore files or an entire file system to a known point in time. Or, you can mount the snapped file system and perform a backup of it to tape (via Tivoli Storage Manager for example). This could help reduce your backup time requirements.

The snapshot remains stable while the original file system is still being modified. You can perform the snapshot operation on a live file system, as it will automatically freeze I/O operations while the image is taken. You may need to briefly quiesce an application for the (very short) duration of the snapshot process. I/O is frozen to provide a greater level of file system consistency in the image. This is no different than if you were to use a similar *copy service* tool such as IBM's Flashcopy on the DS8000.

A snapshot maps its contents to the contents of the source file system. If the source is not modified, the snapshot does not store any of the files in its own physical disk space and has the same content as the original file system. According to the IBM documentation, a snapshot typically needs 2-6% of file system space. Highly active file systems may require around 15% of space. This space is used if a block in the original file system changes. Changed data is then copied to the snapshot.

When a write or delete occurs in the original file system, the affected blocks are copied into the snapshot. If the source is modified, the original contents of the blocks are copied to the disk space area of the snapshot file system. When the snapshot is modified, it either retrieves the data from the original file system (if it is not modified) or it retrieves it from its own disk storage (if the original was changed).

A read of the snapshot will require a lookup to determine whether the block needed should be read from the snapshot or from the original file system. For instance, the read will occur from the snapshot only if the source block has changed since the snapshot was created. If it is unchanged then the read will take place from the original file system.

There are two types of JFS2 snapshots. The first is an *external snapshot* which uses a separate logical volume for the snapshot image. The second type allocates space out of the original file system, this is known as an *internal snapshot*.

An external snapshot can be mounted separately on its own mount point. A file system can use only one type of snapshot method at a time; that is, you cannot take an external snapshot of a file system and then take an internal snapshot of the same file system while the external exists (and vice versa).

It is also important to note that you cannot use internal snapshots unless the file system was enabled to support them at the time of file system creation. So if you want to use snapshots on an existing file system and you did not enable it for internal snapshots, you will need either to recreate the file system with the *snapshot* option or use an external snapshot.

One of the big advantages to using JFS2 snapshots is that we do not need to create a complete copy of all of the data. Nor do we need to recover the entire contents of the data. Thus reducing overhead and saving time.

Using JFS2 snapshots.

In the following examples, I will demonstrate the use of *external* and *internal* JFS2 snapshots.

I needed to perform an upgrade of the lpar2rrd software on one of my AIX systems. Obviously, I wanted to take a backup of the associated file system prior to me applying the update, just in case anything went wrong. I also wanted a way to test the new version of the software and then easily revert to the previous version if I had any issues.

So, before I started, I took a snapshot of the lpar2rrd file system (/lpar2rrd). Of course, I would need to use an *external* snapshot, as I did not create this file system with the *isnapshot=yes* option. To avoid any possible data consistency issues during the upgrade, I shut down the httpd daemon and stopped the lpar2rrd script from running.

The file system in this example is very small, only 256MB in size. It is a regular JFS2 file system in a regular volume group. Snapshots of terabyte file systems are also supported!

```
# df -m /lpar2rrd

Filesystem    MB blocks      Free   %Used    Iused %Iused Mounted on

/dev/lpar2rrdlv    256.00    255.52     1%      10     1% /lpar2rrd
# mount | grep lpar2

/dev/lpar2rrdlv  /lpar2rrd        jfs2   Feb 10 17:44 rw,log=/dev/log01
# grep -p lpar2rrd /etc/filesystems
/lpar2rrd:
        dev             = /dev/lpar2rrdl
        vfs             = jfs2

        log             = /dev/log01
        mount           = true

        account         = false
```

The /lpar2rrd file system contained the lpar2rrd code and the associated RRD data files for each of my managed systems. It also contained a tar zipped file of the new distribution I was about to install:

```
# pwd
/lpar2rrd/data

# ls -ltr
total 184
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A99_p595-3
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A99_p595-2
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A98_p595-1
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN1001C99_p570-1
-rwxr-xr-x    1 lpar2rrd www   90151 Feb 10 17:56 lpar2rrd-dist-2.51.tar.z
```

Creating an external snapshot

I executed the snapshot command to create an external snapshot of the file system. I specified a size of 128MB for the snapshot logical volume (LV). This was half the size of the original (source) logical volume. A new LV was created (fslv08) which contained the external snapshot.

```
# snapshot -o snapfrom=/lpar2rrd -o size=128M
Snapshot for file system /lpar2rrd created on /dev/fslv08
# lsvg -l datavg | tail -2
lpar2rrdlv          jfs2       2       2       1    open/syncd    /lpar2rrd
fslv08              jfs2       1       1       1    open/syncd    N/A
```

To verify the status of the newly created snapshot, I ran the snapshot command with the –q flag. This provided me with information such as the date/time of the image, the location of the image (LV name), the size of the LV and the free space in the image. The `*` identifies the current snapshot.

```
# snapshot -q /lpar2rrd
Snapshots for /lpar2rrd
Current  Location      512-blocks        Free Time
*        /dev/fslv08       262144      261376 Wed Feb 10 18:03:15 CST
2010
```

If you have several large snapshots to manage, then it is a good idea to monitor the usage of your snapshots to determine if they are growing and if they require more disk space. If a snapshot runs out of space, then all the snapshots are invalidated and become unusable.

Another way to check the amount of free space in an external snapshot is to mount the image and use the df command. If an external snapshot needs more space, you can dynamically increase the size of the snapshot logical volume by using the *size* option with the snapshot command. You cannot dynamically reduce the size of a snapshot LV.

```
# snapshot -o size=+1 /dev/fslv08
Snapshot /dev/fslv08 size is now 524288 512-blocks.

# snapshot -o size=-1 /dev/fslv08
snapshot: 0507-562 Cannot reduce the size of a snapshot.
```

To confirm that the snapshot image was intact and that it contained a copy of the entire file system, I mounted the external snapshot to verify the image. I created a separate mount point for the snapshot file system (`/mnt/snapfs`) and then mounted it as a snapshot (with the –o snapshot flag).

```
# mkdir /mnt/snapfs
# mount -v jfs2 -o snapshot /dev/fslv08 /mnt/snapfs
```

The snapshot file system was mounted as a *read-only* snapshot.

```
# mount | grep fslv08
         /dev/fslv08      /mnt/snapfs      jfs2   Feb 10 18:09 ro,snapshot
```

The df command confirmed that the snapped file system was mounted. Using the ls command, I was able to verify that it did indeed contain a copy of the entire source file system.

```
# df -m /mnt/snapfs
Filesystem    MB blocks      Free %Used    Iused %Iused Mounted on
/dev/fslv08      128.00    127.62    1%        -     -  /mnt/snapfs

# ls –ltr /mnt/snapsfs/data
total 184
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A99_p595-3
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A99_p595-2
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A98_p595-1
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN1001C99_p570-1
-rwxr-xr-x    1 lpar2rrd www   90151 Feb 10 17:56 lpar2rrd-dist-2.51.tar.z
```

As the file system was mounted as read-only, I was unable to create a new file or modify an existing one.

```
# cd /mnt/snapfs/data
# touch 1
touch: 0652-046 Cannot create 1.

# rm lpar2rrd-dist-2.51.tar.z
rm: Remove lpar2rrd-dist-2.51.tar.z? y
rm: 0653-609 Cannot remove lpar2rrd-dist-2.51.tar.z.
The file system has read permission only.
```

However, I could copy data out of the file system. This would be an easy way to recover individual files from the snapshot, rather than restoring the entire file system image. If you wish to restore individual files back to their original state, then you can mount the snapshot and then manually copy the files back over.

```
# pwd
/mnt/snapfs/data
# cp -p lpar2rrd-dist-2.51.tar.z /tmp
# ls -ltr /tmp/lpar2rrd-dist-2.51.tar.z
-rwxr-xr-x    1 lpar2rrd www   90151 Feb 10 17:56 /tmp/lpar2rrd-dist-2.51.tar.z
```

At this point, I could perform the lpar2rrd upgrade, knowing that if I had an issue I could recover from my snapshot.

If the snapshot was no longer required, I could simply delete it at a later stage, with the following snapshot command:

```
# cd /
# umount /mnt/snapfs
# snapshot -d /dev/fslv08
```

If I wanted to recover the entire file system from the snapshot, I would execute the rollback command. This reverts a JFS2 file system to a point-in-time snapshot. The original file system must be unmounted before executing the roll back. After the operation, the snapshot is removed (including the associated logical volume).

In this example, you can see that the time stamp on each of the original data directories has changed as part of the lpar2rrd upgrade. In addition, the tar zipped file has been removed.

```
# ls –ltr /lpar2rrd/data
total 0
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 19:02 SN8379A99_p595-3
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 19:02 SN8379A99_p595-2
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 19:02 SN8379A98_p595-1
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 19:02 SN1001C99_p570-1
```

To revert to my previous snapshot and ultimately the older version of lpar2rrd, I will use the rollback command. Along with the source file system (/lpar2rrd), I must also ensure that I have unmounted the external snapshot file system (/mnt/snapfs). To roll back the /lpar2rrd file system, to the external snapshot on logical volume /dev/fslv08, with verbose output (-v), I would enter:

```
# umount /lpar2rrd
/dev/lpar2rrdlv has mounted snapshots which must be unmounted first:
  /mnt/snapfs
umount: 0506-347 Cannot find anything to unmount.

# umount /mnt/snapfs

# umount /lpar2rrd
# rollback -v /lpar2rrd /dev/fslv08
Restoring block 1
Total blocks restored = 32
rmlv: Logical volume fslv08 is removed.

Rollback complete
```

Now I can mount the /lpar2rrd file system again. The data files have been restored to their original state (the date/time stamp is the same as when the images was taken). In addition, the ownership, group and permissions have been preserved.

```
# mount /lpar2rrd
# ls –ltr /lpar2rrd/data
total 184
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A99_p595-3
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A99_p595-2
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN8379A98_p595-1
drwxr-xr-x    2 lpar2rrd www             256 Feb 10 17:46 SN1001C99_p570-1
-rwxr-xr-x    1 lpar2rrd www   90151 Feb 10 17:56 lpar2rrd-dist-2.51.tar.z
```

At this point, I can restart lpar2rrd with the previous version.

Creating an Internal snapshot

To demonstrate the use of an internal JFS2 snapshot, I will create a new file system with the *isnapshot=yes* option.

```
# mklv -t jfs2 -y cglv datavg 1

# crfs -vjfs2 -d cglv -m /cg -A yes -a isnapshot=yes
File system created successfully.
130864 kilobytes total disk space.
New File System size is 262144

# mount /cg

# lsfs -q /cg
Name            Nodename   Mount Pt               VFS   Size    Options    Auto Accounting
/dev/cglv       --         /cg                    jfs2  262144  --         yes  no
  (lv size: 262144, fs size: 262144, block size: 4096, sparse files: yes, inline log: no,
inline log size: 0, EAformat: v2, Quota: no, DMAPI: no, VIX: yes, EFS: no, ISNAPSHOT:
yes)
```

I will also create a 100MB dummy data file for the purposes of this test.

```
# cd /cg

# lmktemp datafile1 100M
datafile1

# ls -ltr
total 204808
drwxr-xr-x    2 root     system          256 Feb 10 19:50 lost+found
-rw-r--r--    1 root     system    104857600 Feb 10 19:53 datafile1

# df -m .
Filesystem    MB blocks      Free %Used    Iused %Iused Mounted on
/dev/cglv        128.00     27.64   79%        6     1% /cg
```

Next I will create an internal snapshot of the /cg file system. The snapshot is named cgsnap1:

```
# snapshot -o snapfrom=/cg -n cgsnap1
Snapshot for file system /cg created on cgsnap1
```

To query the date and time of creation and the snapshot names for the /cg file system, I execute the following command:

```
# snapshot -q /cg
Snapshots for /cg
Current  Name         Time
*        cgsnap1Wed Feb 10 19:55:18 CST 2010
```

A special directory (called .snapshot) is created in the original file system. This contains a *link* to the location of the snapshot image, even though the snapshot directory is read-only (as shown here):

```
# cd /cg/.snapshot/cgsnap1
# touch 1
touch: 0652-046 Cannot create 1.
# rm datafile1
rm: Remove datafile1? y
rm: 0653-609 Cannot remove datafile1.
The file system has read permission only.
```

The .snapshot directory is hidden to avoid unintentional or accidental corruption/deletion of the data. However, I can view data in the snapshot directory if I wish. In this directory, I find another directory named cgsnap1, my snapshot name.

```
# cd /cg/.snapshot
# pwd
/cg/.snapshot

# ls -ltr
total 0
drwxr-xr-x    3 root     system          256 Feb 10 19:55 cgsnap1
```

I can cd to this directory and view the files in the snapshot image.

```
# cd cgsnap1
# ls -ltr
total 204800
drwxr-xr-x    2 root     system          256 Feb 10 19:50 lost+found
-rw-r--r--    1 root     system    104857600 Feb 10 19:53 datafile1
```

If I wanted to recover individual files from an internal snapshot, all I need do is change directory to the appropriate snapshot location and copy the files manually using the cp command.

```
# cd  /cg/.snapshot/cgsnap1
# ls -ltr
total 204800
drwxr-xr-x    2 root     system          256 Feb 10 19:50 lost+found
-rw-r--r--    1 root     system    104857600 Feb 10 19:53 datafile1

# cp -p datafile1 /tmp
# ls -ltr /tmp/datafile1
-rw-r--r--    1 root     system    104857600 Feb 10 19:53 /tmp/datafile1
```

Unlike external snapshots, I do not need to mount the snapshot file system to access the files in the image. Of course, as with any file copy operation, I am always careful to check that a file's ownership, group and permissions are preserved (where appropriate), hence why I use the –p flag with the cp command. Be sure to check these before recovering a file with this method.

What if you wanted to recover an entire file system from an internal snapshot? For example, what if you receive a phone call from a user that goes something like “Whoops, I’ve just zero'ed out my data files. What do I do now?”.

```
# >datafile1
# ls -ltr
total 0
drwxr-xr-x    2 root     system          256 Feb 10 19:50 lost+found
-rw-r--r--    1 root     system            0 Feb 10 21:22
datafile1
```

Well, luckily you created an internal snapshot of the file system. To restore the file system to the required point in time, you can use the rollback command again. To roll back the /cg file system, using the internal snapshot named cgsnap1, execute the following command (note: the original file system must be un-mounted first):

```
# umount /cg
# rollback –v -n cgsnap1 /cg
File system /cg being rolled back to snapshot cgsnap1
Snapshots remaining to rollback: 1
Rebuilding block map state using fsck.

The current volume is: /dev/cglv
Primary superblock is valid.
*** Phase 1 - Initial inode scan
*** Phase 2 - Process remaining directories
*** Phase 3 - Process remaining files
*** Phase 4 - Check and repair inode allocation map
*** Phase 5 - Check and repair block allocation map
Block allocation map is corrupt (FIXED)
Superblock marked dirty because repairs are about to be written.
File system is clean.
Superblock is marked dirty (FIXED)
All observed inconsistencies have been repaired.

Rollback complete
```

And now the data file is restored to its previous state!

```
# mount /cg
# cd /cg
# ls -ltr
total 204800
drwxr-xr-x    2 root     system          256 Feb 10 19:50 lost+found
-rw-r--r--    1 root     system    104857600 Feb 10 19:53 datafile1
```

The snapshot has been removed from the /cg file system.

```
# snapshot -q /cg
/cg has no snapshots.
```

Of course, if the internal snapshot is no longer needed it can simply be deleted.

```
        # snapshot -d -n cgsnap1 /cg
```

Some considerations when using internal snapshots

- When working with an internal snapshot, the very act of creating an image consumes disk space in the existing/original file system. To monitor how much space an internal snapshot is consuming, you need to monitor the size of the original file system using commands like df. If the original file system is running out of space, you may need to remove any unnecessary snapshots or files. You may like to keep several snapshots of a file system, but just keep in mind that this will eat into the space of the original file system and will need to be managed.
- Internal snapshots are preserved when the logredo command runs against a JFS2 file system with an internal snapshot.
- Internal snapshots are removed if the fsck command has to modify a JFS2 file system to repair it.
- If an internal snapshot runs out of space, or if a write to an internal snapshot fails, all internal snapshots for that source file system are marked invalid. Further access to the internal snapshots will fail. These failures write an entry to the AIX error log.

```
# snapshot -q /cg
```

```
Snapshots for /cg
Current  Name         Time
INVALID  cgsnap2      Sun Feb 14 18:48:27 CST 2010
INVALID  cgsnap1      Sun Feb 14 18:50:26 CST 2010
INVALID  cgsnap3      Sun Feb 14 18:51:10 CST 2010

# errpt
IDENTIFIER TIMESTAMP  T C RESOURCE_NAME  DESCRIPTION
96977244   0214185110 P O SYSJ2         Unable to allocate in JFS2 snapshot
F7FA22C9   0214185110 I O SYSJ2         UNABLE TO ALLOCATE SPACE IN FILE SYSTEM


# errpt -aNSYSJ2
---------------------------------------------------------------------------
LABEL:          J2_SNAP_FULL
IDENTIFIER:     96977244

Date/Time:       Sun Feb 14 18:51:18 CST 2010
Sequence Number: 124
Machine Id:      00CED82E7C00
Node Id:         gibaix
Class:           O
Type:            PERM
WPAR:            Global
Resource Name:   SYSJ2

Description
Unable to allocate in JFS2 snapshot

Probable Causes
INSUFFICIENT STORAGE

User Causes
INSUFFICIENT STORAGE MEDIA SPACE

        Recommended Actions
        The snapshots for the JFS2 file system are now invalid.
        Increase size of snapshot storage object and create new snapshot.

Detail Data
Snapshot Major/Minor device number
000A 0016
Snapshot generation number
0000 0004
FILE SYSTEM DEVICE AND MOUNT POINT
/dev/cglv, /cg
```

- Internal snapshots are not separately mountable.
- Internal snapshots are not compatible with AIX releases prior to AIX 6.1. A JFS2 file system created to support internal snapshots cannot be modified on an earlier release of AIX.
- A JFS2 file system with internal snapshots cannot be defragmented.
- The following example will give you an idea of how an internal snapshot can take up file system space.
  First, I'll create a 100M file.

```
# lmktemp datafile1 100M
datafile1
```

The 100M of file system space is used by `datafile1`. The source file system (`/cg`) is 128MB in size, so there is roughly 27MB of free space in the file system now.

```
# df -m .
Filesystem    MB blocks      Free %Used    Iused %Iused Mounted on
/dev/cglv        128.00     27.61   79%        6     1% /cg
```

Now, I'll create an internal snapshot of the file system.

```
# snapshot -o snapfrom=/cg -n cgsnap1
Snapshot for file system /cg created on cgsnap1
```

Notice that very little disk space was use to create the snapshot (i.e., the value under "`Free`" has only decreased marginally).

```
# df -m .
Filesystem    MB blocks      Free %Used    Iused %Iused Mounted on
/dev/cglv        128.00     27.48   79%        7     1% /cg
```

I will now zero out my data file (datafile1). This would normally free up space in the file system but because of the internal snapshot there is very little change in the amount of free space in the file system. This is because all the modified blocks from the source have been copied to the disk in the internal snapshot. Instead of a map to the original data, there are now new blocks with data, consuming disk space in the file system.

```
# >datafile1

# ls -ltr
total 0
drwxr-xr-x    2 root     system          256 Feb 10 19:50 lost+found
-rw-r--r--    1 root     system            0 Feb 11 18:37 datafile1

# df -m .
Filesystem    MB blocks      Free %Used    Iused %Iused Mounted on
/dev/cglv        128.00     27.36   79%        7     1% /cg
```

If I remove the internal snapshot, there is now free space in the file system as expected.

```
# snapshot -d -n cgsnap1 /cg
# df -m .
Filesystem    MB blocks      Free %Used    Iused %Iused Mounted on
/dev/cglv        128.00    127.61    1%        6     1% /cg
```

Summary

JFS2 snapshots are another utility you can deploy alongside other backup tools in your enterprise.

It goes without saying that the use of snapshots is not a replacement for regular backup/restore utilities such as **mksysb**, **savevg**, **Tivoli Storage Manager**, etc. Nor does it protect you from situations where data has been lost due to a hardware or disk drive failure, data corruption, or other potential physical disasters. You still need a reliable backup, restore and disaster recovery plan in place for such events.

My examples are simple and straightforward. The environment is also not complex. If you were planning on using these procedures in your environment with "enterprise class" applications, I'd encourage you to test the procedures thoroughly before implementing them in a production environment. For example, if you needed to use snapshots with a large SAP and Oracle system, this would require extensive testing and possibly consultation with the application vendor to verify and confirm support for this method.

What I have demonstrated here is just one basic example of using JFS2 snapshot technology. There are many other potential uses, such as the **backsnsap** utility. This will create an external

snapshot and then perform a backup of that snapshot to a backup device such as tape. There is also some integration with Tivoli Storage Manager. You can find out more about these and other JFS2 snapshot uses in the links provided under the Resources section.

Related topics

- Learn about JFS2 snapshots
- JFS online backups and JFS2 snapshots
- backsnap command
- AIXchange: Snapshots make file recovery a snap
- Tivoli Storage Manager - AIX configuration considerations prior to performing snapshot-based file backups and archives
- Tivoli Storage Manager - Performing snapshot-based file backup and archive and snapshot-based image backup
- Tivoli Storage Manager - Resolving errors during AIX JFS2 snapshot-based backup-archive and image backup