8 *Managing System Backups*

System backup is an important but sometimes overlooked operation. The unexpected happens occasionallyÐa file is inadvertently deleted, or a power failure interrupts your work in a document. It's a common, but unfortunate, practice to delay creating backups because the procedure can be troublesome and time-consuming. Experienced people (sadder and wiser) will tell you that a regular backup routine is an essential element of system administration. This chapter contains some suggestions to simplify the task.

Backup Strategies

A backup schedule and scheme are essential. How often you do a backupÐthat is, make a copy and store it in a safe placeÐdepends on such factors as how frequently your files change, how much time a backup requires, and how much data you're willing to lose. For example, a full backup of your entire system is thorough, but it requires so much time and storage media that it might not be practical on a daily basis. Since many files change infrequently, it's sensible to do a full backup less often (once a month is common), and do a more frequent backup of those files that change regularly.

Your backup strategy might include several levels of backups, each of which copies different files:

- A *full backup* completely copies all data on your system, although you can omit certain directories or file systems, such as **/tmp**, if you don't want to save the data they contain.

- An *incremental backup* copies only those files that have changed since the last backup. Some programs (such as **dump**) have a built-in incremental backup feature.
- A *partial backup* copies only the files or directories that you specify.

If you're a user on a standalone system, your backup strategy can be as simple as making a copy of your work on a floppy disk each day. In addition, you'll probably want to make a full backup periodically (weekly or monthly).

If you're responsible for administering a network, your backup strategy becomes far more complex. The safest backup strategy is to schedule several levels of backups into an overlapping routine. Here's a good strategy to use:

- Once a month, perform a full backup. It's a good idea to permanently store the media containing these backups in a secure, off-site location.
- Once a week, perform an incremental backup that copies all files that have changed since the most recent monthly backup. You can recycle the media you use for these backups on a monthly basis.
- Once a day, perform an incremental backup that copies all files that have changed since the most recent weekly backup, and perform a separate full backup of administrative information located in **/private/etc** and **/private/adm** (this includes the NetInfo databases and administrative logs and scripts). The media you use for these backups can be recycled weekly.

Warning: Always write a backup to a different tape or disk than the one on which you've stored your most recent backup. This ensures that a failure during a backup today will not destroy yesterday's backup in the process. For the same reason, always check that your most recent backup is readable before disposing of the previous backup.

To save effort, you can use **cron** to perform backups automatically at regular intervals. Incorporate your backup commands into the **cron** scripts **daily**, **weekly**, and **monthly** located in **/usr/adm**. The **cron** utility will execute **/usr/adm/daily** at 2:00 am each day, **/usr/adm/weekly** each Saturday at 3:30 am, and **/usr/adm/monthly** at 5:30 am on the first day of each month.

Using **cron**, your backups can run unattended. Make sure you insert the backup medium into the appropriate drive each evening. The following morning, remove the backup, label it, and file it in a secure location. With careful planning, you can structure your backups to avoid the need for multiple volumes.

What to Back Up

Any data that you can't afford to lose should be backed up. However, some files are more important to back up frequently, either because they change a lot, or because they contain vital information. Give special attention to the following files:

- Documents you've been working onÐIt's a good idea to back up these files nightly, so you don't risk losing more than a day's work. Remember, too, that a printed copy is a form of backup. Although it's not as easy to restore as an electronic file, a printed copy of your work is insurance against total loss.
- Home directoriesÐIf you're only backing up your own work, once a week is probably enough, as long as you're backing up critical files daily. If you're responsible for users on a network, you should back up their home directories more often (probably daily).
- **/private/etc** and **/private/adm**ÐBacking up these directories protects the most important parts of the system databases. For example, these directories contain the administrative information about user accounts, including their encrypted passwords. These directories should be backed up separately and completely to protect the internal consistency of the NetInfo database.

  A NetInfo database is made up of separate files, and you need to make sure they're backed up together at the same time. The **tar** program is a good choice for this task because it lets you copy all files at once. Incremental backups, such as those created with **dump**, might copy the files at different times, introducing inconsistencies. If you're administering a network, the **/private/etc** and **/private/adm** directories should be backed up daily.
- Directories that change oftenÐAny files that change frequently should be backed up frequently. Some directories that might fall into this category are: **/LocalLibrary**, **/LocalApps**, and **/LocalAdmin** (if you have them), **/usr/local**, and **/usr/spool/mail**. Remember to back up any other files or directories you change, such as shared project directories.
- Critical, site-specific filesÐIf there are any other files that you would need to recover in case of a major catastrophe (such as fire or flood), back them up and store them off-site.

Since the system software doesn't change frequently, and takes up a lot of space, you can omit it from your regular backup schedule. If you choose to do this, be sure you have a copy of the system software available, as well as the means to restore the system software from the archived copy.

Ways to Back Up

Backup techniques range from the simple to the complex. For example, you can do a small backup simply by copying a file or directory to another computer on the network or to a second storage device (such as an optical disk, second hard drive, or floppy disk).

To do more sophisticated backups, you can use a number of programs. Each has its advantages and disadvantages. This section discusses six: **dump** and **restore** (used together), **rdump** and **rrestore** (used together), **tar**, and **cpio**.

Here are some things to consider in your backup procedures:

- Log in as **root**, a member of the **operator** group (best choice), or the owner of the files you're backing up or restoring. When you're restoring, make sure you have write permission in the directory that will receive the backed-up material. It's a good idea to add the accounts of those who do backups to the **operator** group so they can create backups without becoming **root**. Remember that **root** might not have access to remote NFS directories.
- Back up the most important data first. If something goes wrong during the backup, or if your backup media runs out of space, you can then be sure that the most important data is safe. Typically, you should back up user data before system data, since user files change more frequently. However, the system files in ***/etc*** change frequently and are critically important, so consider backing up these files first.
- Perform backups in single-user mode, if possible, or at a time when no one is using the system. (It's not a good idea for a file to be modified while it's being backed up.) Usually, backups are done at night.
- Label your backup media carefully. A label should contain the date and time of the backup, a list of the contents, and the type of backup done. This descriptive labeling can save you time when data needs to be

retrieved. It's also a good idea to keep a written log documenting what you've done. Keep in mind that you may need this information when the system is down, so don't store it on-line.

- Create a file containing a table of contents of the backup. This should be the first file you write to the tape.
- Set the write protection on your media to protect against accidental overwrites. Be sure to turn off the write protection before you reuse the media.
- Store your media in a secure location. Make sure that only authorized people have access to it.

The dump and restore Programs

The **dump** program gives you a way to back up your files using an incremental backup approach. An incremental backup only copies files that have been modified since the last backup. Since some data on the system rarely changes, you can greatly reduce the amount of data you have to back up by copying only what has changed. The **restore** program lets you retrieve files you've backed up with **dump**.

Important: It's recommended that you run **dump** and **restore** from an account *other* than **root** that's a member of the **operator** group. This precaution protects you from accidentally overwriting your disk by making a typing error when executing **dump** or **restore**.

Advantages of dump and restore:

- Easy way to store and retrieve a large amount of data
- Can perform and track interactive backups and restores
- Can perform incremental backups
- Fast

Disadvantages of dump and restore:

- Can only back up entire file systems; can't be used to back up a single directory or file

Using dump

The **dump** program keeps track of how extensive a backup should be by assigning each backup a level number from 0 to 9. A level 0 **dump** copies the entire file system, and each subsequent **dump** level copies those files that have been modified since the most recent **dump** with a lower level number. For example, a level 5 **dump** copies all changed files since the most recent **dump** with a level of 4 or lower. If executed with the **u** option, **dump** keeps a record of the date, time, and numeric level at which a file system was backed up, in the file **/etc/dumpdates**.

A simple backup strategy is to start with a complete level 0 **dump** and then make incremental backups each day.

Warning: Before doing a level 0 **dump**, it's important to run **fsck** to be certain you're copying a clean file system. Skipping this step risks dangerous file system corruption. Always run **fsck** on an unmounted file system. In addition, it's recommended that the file system be unmounted when doing a level 0 **dump**, to make sure you have a consistent and complete backup. If you're backing up the root file system, you should do this in single-user mode. For more information on **fsck**, see the UNIX manual page for **fsck** and Chapter 9, ªSystem Startup and Shutdown.º

Examples Using dump

1. To make a complete level 0 backup, enter a command similar to the following:

   ```
   dump 0u /dev/rsd0a
   ```

   This creates a full backup of **/dev/rsd0a**, writing the output to the default backup device (**/dev/rxt0**, an external SCSI tape drive).

Warning: When making a level 0 **dump**, *never* overwrite your most recent backup. Instead, set it aside and create a new one on a second tape. This protects against your only level 0 backup being damaged (for example, if the power should fail while you're doing a backup).

2. To create a daily, weekly, or monthly backup, enter a command similar to the following (replace *9* with the appropriate backup level and *filesystem* with the appropriate device name):

   ```
   dump 9u filesystem
   ```

   Remember, you can put your backup commands in a **cron** script to have them run automatically. Be sure to

label your **dump** tapes carefully.

Note: The specific level numbers used for your backups aren't significant, as long as you use three different numbers and are consistent. You can use level 1 for monthly backups, 2 for weekly backups, and 9 for daily backups, or you could use 1 for monthly, 4 for weekly, and 7 for daily.

3. If you want to specify a different output file or device, use the **f** option:

   ```
   dump 9uf /Dumps/SD1a-9.dump /dev/rsd1a
   ```

   This example creates a level 9 backup of **/dev/rsd1a**, writing it to the file **/Dumps/SD1a-9.dump**. This file might be located on a removable disk (assuming the removable disk is mounted on **/Dumps**). Here, the **.dump** suffix is used to help identify the file as a **dump** backup.

4. To conserve space on your backup media, you can compress the output:

   ```
   dump 9uf - /dev/rsd0b | compress > /Dumps/SD0b-9.dump.Z
   ```

   Using ª-º as the output file sends the output to standard out (**stdout**). The output from **dump** is then used as input for the **compress** command. Finally, the output from **compress** is redirected to the file **/Dumps/SD0b-9.dump.Z**. (The **.Z** suffix is a convention indicating the file is compressed.) Use the **uncompress** or **zcat** command to recover the original, full-size **dump** file.

5. To decrease the amount of space required to do a full backup of the root file system, you can omit the system software from the normal full backup (since it should never change). Make sure you have some means of installing the system software, then enter the following command:

   ```
   dump 0uf /dev/null /dev/rsd0a
   ```

Warning: If you choose to do this, you must create this ªphonyº level 0 **dump** *before* you do anything else with a new system (or immediately after installing the system software, if you're using another disk as your boot disk).

This command updates **/etc/dumpdates** to indicate that a level 0 **dump** has been performed without actually creating a backup. (The output is sent to **/dev/null**, the *null device*, also called the ªbit bucket.º Any output sent here is discarded.) From now on, use a level 1 **dump** to make all your full backups of the root file system. To be consistent, you can also use a level 1 **dump** to do the full backups of your other file systems, provided that you never do a level 0 **dump** of them.

For a complete list and description of the arguments available with **dump**, see the UNIX manual page for **dump**.

Using restore

When you want to retrieve the data you've backed up with **dump**, use **restore**. The **restore** program copies data from the **dump** file to your current directory. Like **dump**, **restore** will use an external tape drive (**/dev/rxt0**) by default, but you can override this with the **f** option.

Warning: When using **restore**, be extremely careful that you don't inadvertently overwrite existing data. Files retrieved with **restore** are written into the current working directory. Have an empty directory ready to receive the restored data. *Never* restore data into a directory that already contains files unless you're certain of the outcome. The **i** option to **restore**, described later in this section, can be particularly helpful.

You must always use **restore** with a key argument. This key determines exactly what **restore** will do. Here are some of the most important key arguments:

- **r**ÐReads the entire **dump** file into the current directory. This key should *only* be used to restore a complete **dump** file onto a clear file system or to restore an incremental **dump** file after restoring a full level 0 backup.
- **x** *filename*ÐExtracts the specified file or directory from a **dump** file. Note that **restore** must create all the directories leading to the file before restoring the actual file. Keep this in mind when planning where you restore the filesÐbe ready to restore a directory hierarchy, not just an individual file. It's a good idea to restore into a temporary directory, then copy or move the restored files into their permanent location.
- **t** *[filename]*ÐDisplays a table of contents for the **dump** file. With the *filename* argument, displays the table of contents for the specified file. Without the *filename* argument, displays the contents for the entire backup.
- **i**ÐPerforms the restore interactively. This is particularly useful for restoring just part of a backup, and it's a good way to see what's in the **dump** file. An interactive restore uses a set of commands that you enter into an interface similar to a shell.

Examples Using restore

1. Here's an annotated example of an interactive **restore** session:

```
mkdir /new
cd /new
/etc/restore if /dev/rxt0
restore > ls                                    Execute a command (ls) at the restore prompt.
.:
./                  LocalAdmin/         files/              me/
../                 LocalApps/          fun.eps             mnta/
.NeXT/              LocalDeveloper/     bin/                odmach
.cshrc              LocalLibrary/       cores/              private/
.hidden             Net/                dev/                savem
.login              NextAdmin/          etc/                sdmach
.profile            NextApps/           lib/                tmp/
.rhosts             NextDeveloper/      lost+found/         usr/
.alias              NextLibrary/        mach@

restore > cd /files                             Change to a particular directory on the tape and
restore > ls                                    list its contents.
./files:
 bin/

restore > cd /bin
restore > ls
./files/bin:
 sh

restore > add sh                                Add the file sh to the list of files to be restored.

restore > extract                               Restore all the files on the list from
                                                the dump tape.
You have not read any tapes yet.
Unless you know which volume your file(s) are on you should start
with the last volume and work towards towards the first.
Specify next volume #: 1                        Specify the tape volume from which to start the
                                                restore.

set owner/mode for '.'? [yn] y                  Set the ownership to the user performing the
                                                restore and permission to their defaults.
```

`restore > quit`	Exit the program.

2. This example does a full restore from the backup to a newly initialized disk (useful if you're upgrading to a new disk):

disk -i */dev/rsd1a*	Initialize the new disk.
mount */dev/sd1a /mnt*	Mount a file system on the disk.
cd */mnt*	Change to the mounted directory.
restore rf */dev/rst0*	Restore the entire contents of the dump file.
rm *restoresymtab*	Delete a temporary file left behind by **restore**.

3. The following command does an interactive restore of a compressed backup:

```
zcat /Dumps/SD0a-9.dump.Z | restore if -
```

For more information on **restore** and its many arguments, see the UNIX manual page for **restore**.

The rdump and rrestore Programs

The **rdump** and **rrestore** programs work much like **dump** and **restore**, except that they're used to back up a file system over the network to a remote device (such as a shared tape drive).

Advantages of rdump and rrestore:

- Can be more convenient than **dump**, since you can use a single tape drive to back up and restore all the computers on a network
- Same as **dump** and **restore**

Disadvantages of rdump and rrestore:

- Same as **dump** and **restore**

Note: To use **rdump** to do a backup of a disk attached to a remote system, you must have permission to execute the **rsh** command as **root** on the computer that has the target output device attached to it. This permission is given by making an entry for your remote computer in the **/.rhosts** file. For example, if you're making a backup of a disk attached to the computer **bilbo** and will be sending the output file to a disk attached to

the computer **gandalf**, you need an entry for **bilbo** in the **/.rhosts** file on **gandalf**. For more information, see the UNIX manual page for **.rhosts**.

You use **rdump** just as you would **dump**, except that you must always use the **f** option to specify the remote device. The **f** option is followed by an argument in the form *host*:*device*.

To restore data across a network that has been copied using **rdump**, use **rrestore**. The **rrestore** program operates almost identically to **restore**, except that you must always specify the remote device from which you are restoring the data. To do this, use the **f** key followed by an argument in the form *host*:*device*. You are then restoring data from the drive *device* on the computer *host*. (Again, make sure you have an appropriate entry in the **/.rhosts** file on the remote computer.)

Like **restore**, **rrestore** always restores data to the current directory. Consequently, before running **rrestore**, you need to log into the computer to which you're restoring data and change to the appropriate directory. If you're not sitting at the computer that will receive the data, you can log in using **rlogin** or **telnet**. For more information, see the UNIX manual pages for **rlogin** and **telnet**.

Examples Using rdump and rrestore

1. Here's an example of a command that writes an incremental backup of **/dev/rsd2a** to the tape drive **/dev/nrst0** on the host **tserver**:

   ```
   rdump 9uf tserver:/dev/nrst0 /dev/rsd2a
   ```

2. This example interactively restores from the tape in drive **/dev/rst0** on **tserver**:

   ```
   rrestore if tserver:/dev/rst0
   ```

For more information on **rdump** and **rrestore**, see the UNIX manual pages for **rdump**, **dump**, **rrestore**, and **restore**.

The tar Program

The **tar** program takes multiple files and directories and stores them as a single large archive file. Its actions are controlled by a key argument. You can also specify file or directory names indicating which files to copy or

restore.

Advantages of tar:

- Can make copies of individual files and directories
- Fast
- Compatible with non-UNIX file systems (for example, you can use **tar** to copy from a DOS diskette, and then restore the data onto another DOS diskette)

Disadvantages of tar:

- Cannot perform more complex operations such as incremental backups
- Restricts file pathnames to 100 characters

Warning: Since **tar** can't handle multiple volumes, it should not be used if the material you're backing up doesn't fit on a single storage volume. Also, **tar** doesn't copy files with names longer than 100 characters. This is of particular note on NeXT computers because the names of some files can be very long.

The **tar** command can be used with a variety of keys to control its operation. Here are some of the most frequently used:

- **c** *filename*ÐCreates a new **tar** file and starts writing from its beginning. (By default, **tar** will read and write an external tape drive, **/dev/rxt0**.) If you specify a directory to copy, **tar** copies the entire directory structure, not just the top level.
- **x** *filename*ÐExtracts the named file from the **tar** file. If you omit the *filename* argument, **tar** will restore the entire contents of the **tar** file. Be careful not to overwrite existing data with this key.
- **p**ÐPreserves the permissions of extracted files.
- **t**ÐLists the contents of the **tar** file without actually extracting the files.
- **h**ÐCauses **tar** to follow symbolic links (it will copy the file being pointed to, not just the link file).
- **v**ÐPrints the name of each file as **tar** copies it. You can combine this with the **t** option to see even more information about the files.

When making an archive using **tar c**, you can use either a *relative* or *absolute* pathname to specify the file or directory to be copied. If you use a relative pathname (that is, any path that doesn't begin with a ª/º), you can later restore the files into any directory. If you use an absolute pathname (that is, a fully qualified pathname beginning with ª/º), **tar** will *only* restore the copied files into the directory from which you originally copied them. In fact, **tar** restores all the directories in the path leading to the file (even if it has to create them) before restoring the actual file.

Note: To provide maximum flexibility when you restore files, it's highly recommended that you use relative pathnames when creating **tar** files.

Examples Using tar

1. The following example illustrates how **tar** is used. Note that you can specify more than one file or directory on the command line. Start by changing to the directory from which you're going to copy the file or directory:

   ```
   cd /
   tar c ./etc ./adm
   ```

   This command copies the directories **./etc** and **./adm** onto the default device **/dev/rxt0**. Note that relative pathnames (beginning with ª./º) are used to identify the directories to be copied.

2. To restore the entire contents of the archive, change to the directory where you want to restore them and enter the following:

   ```
   tar xp
   ```

   Since you used a relative pathname when you made the backup copy, **tar** restores the backup into your current working directory. The **p** option preserves file permissions. Note that though **tar** attempts to restore the correct file permissions, it can't guarantee this.

For more information, see the UNIX manual page for **tar**.

The cpio Program

The name **cpio** stands for copy input to output. This command copies files into and out of a **cpio** archive. It uses

the standard input as its source of file names and the standard output as the archive output. This means that you can combine **cpio** with standard UNIX commands (such as **find** or **ls**) to create a wide variety of specialized backups. For example, you can combine **find** and **cpio** to create a backup containing all the files owned by a particular user that were last modified over six weeks ago.

Advantages of cpio:

- Can be used to copy and restore single files
- Can perform incremental backups
- Can be used to copy files meeting any condition (age, ownership, size, and so forth)

Disadvantages of cpio:

- Has a cryptic command syntax
- Doesn't make effective use of storage space
- Restricts pathnames to 128 characters
- Can't be used for backing up directories or files with symbolic links

The **cpio** program uses two options for backups:

- **-o** copies out an archive (to create a backup).
- **-i** copies in an archive (to restore from a backup).

Each of these primary options can be combined with additional options to further modify what **cpio** does. For a complete description of these options, see the UNIX manual page for **cpio**.

Unlike **dump** or **tar**, the **cpio** command isn't used by itself to create a backup copy. Instead, it gets a list of which files are to be copied from another UNIX command (typically **find** or **ls**). The arguments used with these UNIX commands determine which files and directories **cpio** will copy.

Examples Using cpio

1. In the following example, **cpio -o** copies all files in the current directory into an archive. Notice how the output of the **ls** command becomes the input to **cpio**:

   ```
   ls | cpio -oB > /dev/rxt0
   ```

Important: When sending output to tape devices (such as **/dev/rxt0**), be sure to use the **B** option, which causes **cpio** to use 5120 byte records. You must also use this option to restore the archive.

2. Here's a command that finds every file in **/NeXTLibrary/Documentation** that is not NFS-mounted, and then archives it:

   ```
   cd /NeXTLibrary/Documentation
   find . -depth -fstype 4.3 -prune -print | cpio -oB > /dev/rxt0
   ```

   For information about the **find** command, see the UNIX manual page.

3. To restore an entire archive, enter a command such as:

   ```
   cpio -idB < /dev/rxt0
   ```

   The **-d** option instructs **cpio** to create directories as needed during the restore.

4. The following command lists the contents of an archive in the long format listing.

   ```
   cpio -ivt < /myfile
   ```

   This command combines **cpio -i** with the **-t** option, which lists the table of contents of the input archive, and the **-v** option (for a verbose listing of file names, similar to the output of **ls -l**).

For more information about **cpio** and its use, see the UNIX manual pages for **cpio**, **find**, and **ls**.

General Tips

The biggest cause of concern when doing backups is the potential for human error. It's easy to overwrite valuable data if you're not paying attention. It's also easy to overlook making a backup, and live to regret it later. Here are some ways to avoid trouble:

- Be sure to have a systematic backup routine and stick to it.
- Check before performing a restore to make sure you're not going to overwrite important files as you read data

back onto the disk. The riskiest commands to use in this regard are **restore r** (remember that **restore** always copies into the current directory), **rrestore r**, **tar x**, and **cpio -i**.

- Be especially aware of the hazards of typing errors when using backup programs such as **dump** and **restore**, since you can overwrite your disk. Try to avoid doing backups while logged in as **root**, if possible.
- If you're making a complete backup and know that you'll definitely be restoring from the backup later (for example, if you're making a backup as a preliminary to replacing a disk), it's best to do the backup twice. This protects against data corruption on one of the copies. It's also a good idea to verify that the backup was successful by reading the contents of the backup tape or disk.
- Always label your backup copies carefully to avoid confusing them, and store them in a safe place.
- If users store valuable data on local disks not available to the usual server backup procedures, be sure to set up a backup routine for this data, or encourage the users to do so.
- Be sure media write protection is turned off before attempting a backup.

Troubleshooting

What follows is a list of the error messages you're most likely to encounter with each of the backup programs.

Messages from dump and rdump

The following message appears if there's no tape drive on the computer you've specified to receive the data, or if there's an error in the way you specified the tape drive:

```
DUMP:NEEDS ATTENTION: Cannot open tape. Do you want to retry the open?
```

If the computer you're using lacks an **/etc/fstab** file, you see the following message:

```
DUMP: Can't open /etc/fstab for dump table information.
```

Messages from rdump and rrestore

If you don't have remote access permission on either the system you're backing up or the tape drive server, you'll get a message such as the following:

```
Permission denied.
```

If you receive such a message, make appropriate entries for your system in the ***/.rhosts*** files on the systems denying access.

Messages from tar

A message such as the following appears if there is no tape drive on the computer you've specified to receive the data (using **tar c**), or if there's an error in the way you specified the tape drive:

```
/dev/rxt0: No such tape or address.
```

If you don't have **root** permission or don't belong to the **operator** group, you'll see a message such as the following:

```
Cannot open /dev/rsd0a
```

Messages from cpio

The **cpio** program has trouble with pathnames longer than 128 characters. If you exceed this length in a directory or file you are copying, **cpio** responds with a message such as:

```
<toolongpathname>?
2936 blocks
```

The **find** command is frequently paired with **cpio**. The **-ncpio** option can't be used with **find**. If you do, you'll get a message such as the following:

```
find: bad option <-ncpio>
```