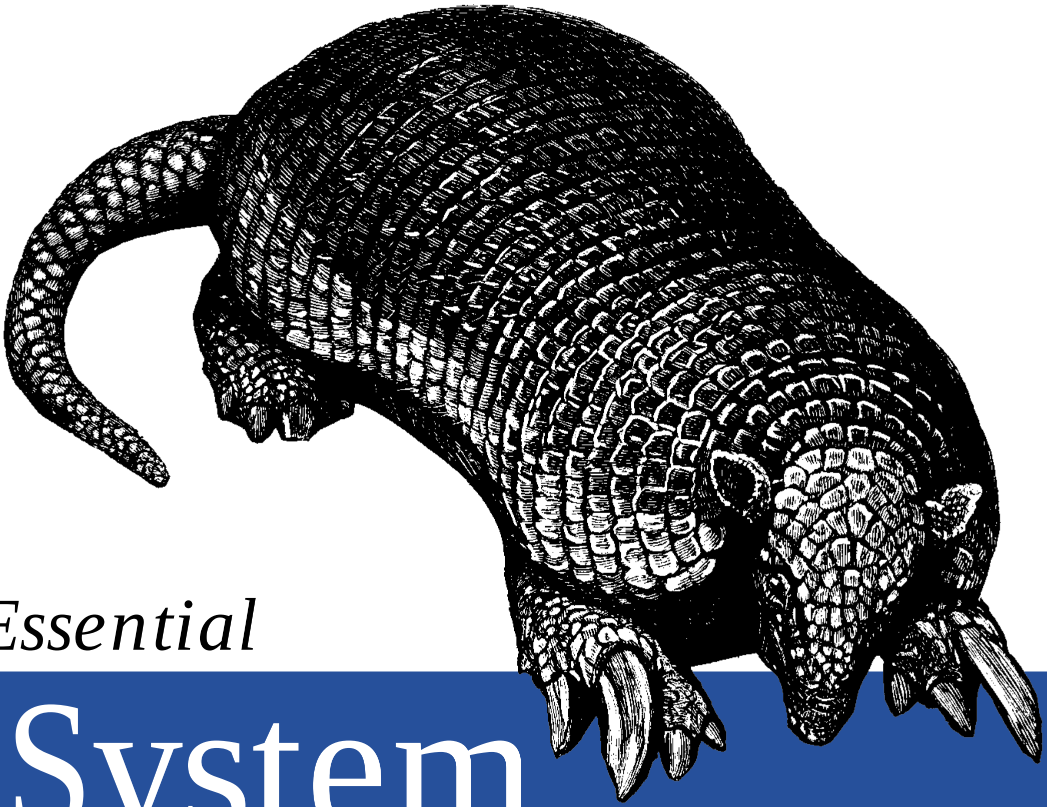
Tools and Techniques for Linux and Unix Administration

3rd Edition
Revised & Updated



Essential System Administration

O'REILLY®

Æleen Frisch

THIRD EDITION

Essential System Administration

Æleen Frisch

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

CHAPTER 11

Backup and Restore

Every user of any computer figures out sooner or later that files are occasionally lost. These losses have many causes: users may delete their own files accidentally, a bug can cause a program to corrupt its data file, a hardware failure may ruin an entire disk, and so on. The damage resulting from these losses can range from minor to expansive and can be very time-consuming to fix. To ensure against loss, one primary responsibility of a system administrator is planning and implementing a backup system that periodically copies all files on the system to some other location. It is also the administrator's responsibility to see that backups are performed in a timely manner and that backup tapes (and other media) are stored safely and securely. This chapter will begin by discussing backup strategies and options and then turn to the tools that Unix systems provide for making them.

An excellent reference work about backups on Unix systems is *Unix Backup and Recovery* by W. Curtis Preston (O'Reilly & Associates). It covers the topics we are discussing here in complete detail and also covers material beyond the scope of this book (e.g., backing up and restoring databases).

Planning for Disasters and Everyday Needs

Developing an effective backup strategy is an ongoing process. You usually inherit something when you take over an existing system and start out doing the same thing you've always done when you get a new system. This may work for a while, but I've seen companies try to retain their centralized, hordes-of-operators–based backup policies after they switched from a computer room full of mainframes to a building full of workstations. Such an attempt is ultimately as comical as it is heroic, but it all too often ends up only in despair, with no viable policy ever replacing the outdated one. The time to develop a good backup strategy is right now, starting from however you are approaching things at the moment.

Basically, backups are insurance. They represent time expended in an effort to prevent future losses. The time required for any backup strategy must be weighed against the decrease in productivity, schedule slippage, and so on if the files are needed but are not available. The overall requirement of any backup plan is that it be able to restore the entire system—or group of systems—within an acceptable amount of time in the event of a large-scale failure. At the same time, a backup plan should not sacrifice too much in the way of convenience, either in what it takes to get the backup done or how easy it is to restore one or two files when a user deletes them accidentally. The approaches one might take when considering only disaster recovery or only day-to-day convenience in isolation are often very different, and the final backup plan will need to take both of them into account (and will accordingly reflect the tension between them).

There are many factors to consider in developing a backup plan. The following questions are among the most important:

What files need to be backed up? The simplest answer is, of course, *everything*, and while everything but scratch files and directories needs to be saved somewhere, it doesn't all have to be saved as part of the system backups. For example, when the operating system has been delivered on CD-ROM, there is really no need to back up the system files, although you may choose do so anyway for reasons of convenience.

Where are these files? This question involves both where the important files are within the filesystem and which systems hold the most important data.

Who will back up the files? The answer may depend on where the files are. For example, many sites assign the backup responsibility for server systems to the system administrator(s) but make users responsible for files that they keep on their workstation's local disks. This may or may not be a good idea, depending on whether or not all of the important files really get backed up.

Where, when, and under what conditions should backups be performed? Where refers to the computer system on which the backup will be performed; this need not necessarily be the same as the system where the files are physically located. Similarly, in an ideal world, all backups would be performed after hours on unmounted filesystems. That's not always practical in the real world, however.

How often do these files change? This information will help you decide both when and how often to perform backups and the type of schedule to implement. For example, if your system supports a large, ongoing development project, the files on it are likely to change very frequently and will need to be backed up at least daily and probably after hours. On the other hand, if the only volatile file on your system is a large database, its filesystem might need to backed up several times every day while the other filesystems on the same system would be backed up only once a week.*

* In actual fact, a database is often backed up using a facility provided by the software vendor, but you get the idea here.

How quickly does an important missing or damaged file need to be restored? Since backups protect against both widespread and isolated file loss, the timeframe in which key files need to be back online needs to be taken into account. The number of key files, how widely spread they are throughout a filesystem (or network), and how large they are will also influence matters. Your system may only have one irreplaceable file, but you'll need to plan very differently depending on whether it is 1 KB or 1 GB in size. Note that losing even a single 1 KB file can wreak havoc if it's the license file without which the central application program won't run.

How long do we need to retain this data? Backups protect current data from accidents. As such, they are normally needed—or useful—only for a relatively short period (months or a year or two) In contrast, most sites also need to create permanent archives of important "point-in-time" data, for example, the software and data used to prepare a tax return. These need to be saved for an indefinite period: many years or even decades. While the requirements are similar, the goals are different enough that you are unlikely to be able to rely on your regular backups for archival purposes. Thinking about this kind of data and how to create and store it must be part of every effective backup plan.

Where should the backup media be stored? Recent backups are generally kept close to the computer for quick restoration. Long-term backups and archives should be stored in a secure offsite location.

Where will the data be restored? Will the backup files be used only on the system from which they were made, or is there an expectation that they could be restored to a different system in an emergency? If multisystem compatibility is ever important, it needs to be taken into account in designing the backup and recovery plan. For example, you might need to ensure that any compression scheme in use on one system can be decoded by the other target systems (or avoid using any vendor-specific formats). Other examples of this sort of issue include access control list data that might be backed up along with files and backups of a filesystem from a system that is larger than the maximum filesystem size on the target system.

Backing up Active Filesystems

Virtually all Unix documentation recommends that filesystems be unmounted before a backup is performed (except for the root filesystem). This recommendation is rarely followed, and in practice, backups can be performed on mounted filesystems. However, you need to make users aware that open files are not always backed up correctly. It is also true that there are circumstances in which events in an active filesystem can cause some files or even the entire backup archive itself to be corrupt. We will consider those that are relevant to the various available backup programs as we discuss them.

Backup Capacity Planning

Once you have gathered all the data about what needs to be backed up and the resources available for doing so, a procedure like the following can be used to develop the detailed backup plan itself:

1. Begin by specifying an ideal backup schedule without considering any of the constraints imposed by your actual situation. List what data you would like to be backed up, how often it needs to be backed up, and what subdivisions of the total amount make sense.
2. Now compare that ideal schedule to what is actually possible in your environment, taking the following points into consideration:
    - When the data is available to be backed up: backing up open files is always problematic—the best you can hope for is to get an uncorrupted snapshot of the state of the file at the instant that the backup is made—so, ideally, backups should be performed on idle systems. This usually translates to after normal working hours.
    - How many tape drives (or other backup devices) are available to perform backups at those times and their maximum capacities and transfer rates: in order to determine the latter, you can start with the manufacturer's specifications for the device, but you will also want to run some timing tests of your own under actual conditions to determine realistic transfer rates that take into account the system loads, network I/O rates, and other factors in your environment. You will also need to take into account whether all the data is accessible to every backup device or not.

    At this point (as with any aspect of capacity planning), there is no substitute for doing the math. Let's consider a simple example: a site has 180 GB of data that all needs to be backed up once a week, and there are 3 tape drives available for backups (assume that all of the data is accessible to every drive). Ideally, backups should be performed only on week nights between midnight and 6 A.M. In order to get everything done, each tape drive will have to back up 60 GB of data in the 30 hours that the data is available. That means that each tape drive must write 2 GB of data per hour (333 KB/sec) to tape.

    This is within the capabilities of current tape drives when writing local data.[*] However, much of the data in our example is distributed across a network, so there is a chance that data might not be available at a fast enough rate to sustain the tape drive's top speed. Some backup programs also pause when they encounter an open file, giving it a chance to close (30 seconds is a typical wait period); when there are a lot of open files in a backup set, this can substantially increase how long the backup takes to complete.

[*] In practice, of course, you would also need an auto loading tape device (or someone to change tapes in the middle of the night).

In addition, we have not made any allowances for performing incremental backups (discussed below) between full backups. Thus, this example situation seems to strain the available resources.

3. Make modifications to the plan to take into account the constraints of your environment. Our example site is cutting things a bit too close for comfort, but they have several options for addressing this:
   - Adding additional backup hardware, in this case, a fourth tape drive.
   - Decreasing the amount of data to be backed up or the backup frequency: for example, they could perform full backups only every two weeks for some or all of the data.
   - Increasing the amount of time available/used for backups (for example, performing some backups on weekends or doing incremental backups during the early evening hours).
   - Staging backups to disk. This scheme writes the backup archives to a dedicated storage area. The files can then be written to tape at any subsequent time. Disks are also faster than tape drives, so this method also takes less time than directly writing to tape. It does, of course, require that sufficient disk space be available to store the archives.
4. Test and refine the backup plan. Actually trying it out will frequently reveal factors that your on-paper planning has failed to consider.
5. Review the backup plan on a periodic basis to determine if it is still the best solution to your site's backup needs.

Backup Strategies

The simplest and most thorough backup scheme is to copy all the files on a system to tape or other backup media. A *full backup* does just that, including every file within a designated set of files, often defined as those on a single computer system or a single disk partition.*

Full backups are time-consuming and can be unwieldy; restoring a single file from a large backup spanning multiple tapes is often inconvenient, and when files are not changing very often, the time taken to complete a full backup may not be justified by the number of new files that are actually being saved. On the other hand, if files are changing very rapidly, and 50 users will be unable to work if some of them are lost, or when the amount of time a backup takes to complete is not an issue, then a full backup might be reasonable even every day.

* For the purposes of this discussion, I'll focus on per-disk partition backups, but keep in mind that this is not the only reasonable way of organizing things. I'll also refer to "backup tapes" most of the time in this chapter. In most cases, however, what I'll be saying will apply equally well to other backup media.

Incremental backups are usually done more frequently. In an incremental backup, the system copies only those files that have been changed since some previous backup. Incrementals are used when full backups are large and only a small amount of the data changes within the course of, say, one day. In such cases, backing up only the changed files saves a noticeable amount of time over performing a full backup.

Some Unix backup programs use the concept of a *backup level* to distinguish different kinds of backups. Each backup type has a level number assigned to it; by definition, a full backup is level 0. Backing up the system at any level means saving all the files that have changed since the last backup at the previous level. Thus, a level 1 backup saves all the files that have changed since the last full (level 0) backup; a level 2 backup saves all the files that have been changed since the last level 1 backup, and so on.*

A typical backup strategy using multiple levels is to perform a full backup at the beginning of each week, and then perform a level 1 backup (all files that have changed since the full backup) each day. The following weekly backup schedule summarizes one implementation of this plan:

> *Monday:* Level 0 (full)
> *Tuesday–Friday:* Level 1 (incremental)

A seven-day version of this approach is easy to construct.

The primary advantage of this plan is that only two sets of backup media are needed to restore the complete filesystem (the full backup and the incremental). Its main disadvantage is that the daily backups will gradually grow and, if the system is very active, may approach the size of the full backup set by the end of the week.

A popular monthly plan for sites with very active systems might look something like this:

> *First Monday:* Level 0 (full)
> *All other Mondays:* Level 1 (weekly incremental to previous Level 0)
> *Tuesday–Friday:* Level 2 (daily incremental to previous Level 1)

This plan will require three sets of backup media to do a complete restore (the most recent backup of each type).

In deciding on a backup plan, take into account how the system is used. The most heavily used portions of the filesystem may need to be backed up more often than the other parts (such as the root filesystem, which contains standard Unix programs and files and which therefore rarely changes). A few parts of the system (like */tmp*) need never be backed up. You may want to create some additional filesystems that

* Not all backup commands explicitly use level numbers, but the concept is valid for and can be implemented with any of the available tools, provided you are willing to do some of the record keeping yourself (by hand or by script).

will never be backed up; anyone using them would be responsible for backing up his own files.

You should also consider performing a full backup—whether the schedule calls for it or not—before you make significant changes to the system, such as building a new kernel, adding a new application package, or installing a new version of the operating system. This may be one of the few times that the root filesystem gets backed up, but if you ever have a problem with your system disk, you will find it well worth the effort when you can avoid a significant amount of reconfiguration.

Unattended backups

The worst part of doing backups is sitting around waiting for them to finish. Unattended backups solve this problem for some sites. If the backup will fit on a single tape, one approach is to leave a tape in the drive when you leave for the day, have the backup command run automatically by `cron` during the night, and pick up the tape the next morning.

Sometimes, however, unattended backups can be a security risk; don't use them if untrusted users have physical access to the tape drive or other backup device and thus could steal the media itself. Backups needed to be protected as strongly as the most secure file on the system.

Similarly, don't do unattended backups when you can't trust users not to accidentally or deliberately write over the tape or other rewriteable media (ejecting the tape after the backup is completed sometimes prevents this, but not always). You also won't be able to use them if the backup device is in heavy use and can't be tied up by the backup for the entire night.

Data verification

In many cases, backups can simply be written to media, and the media can go directly to its designated storage location. This practice is fine as long as you are 100% confident in the reliability of your backup devices and media. In other cases, data verification is a good idea.

Data verification consists of a second pass through the backed-up data, in which each file is compared to the version on disk, ensuring that the file was backed up correctly. It also verifies that the media itself is readable.

Some sites will choose to verify the data on all backups. All sites should perform verification operations on at least a periodic basis for all of their backup devices. In addition, as they age and wear out, many devices begin to produce media that can only be successfully read in the drive that produced it. If you need backups that will be readable by devices or systems other than the one that originally wrote on the physical media, you should also periodically verify the backups' readability by examining them on the target devices and systems.

Storing backup media

Properly storing the backup tapes, diskettes, or other media once you've written them is an important part of any backup plan. Here are some things to keep in mind when deciding where to store your backup media:

Know where things are. Having designated storage locations for backups makes finding the right one quickly much more likely. It is also important that anyone who might need to do a restore knows where the media are kept (you will want to take a vacation occasionally). Installation CDs, bootable recovery tapes, boot diskettes, and the like also ought to be kept in a specific location known to those people who may need them. I can assure you from personal experience that a system failure is much more unpleasant when you have to dig through boxes of tapes or piles of CDs looking for the right one before you can even attempt to fix whatever's wrong with the system.

Another aspect of knowing where things are concerns figuring out what tape holds the file that you need to restore. Planning for this involves making records of backup contents, which is discussed later in this chapter.

Make routine restorations easy. Backups should be stored close enough to the computer so that you can quickly restore a lost file, and tapes should be labeled sufficiently well so that you can find the ones you need.

Ideally, you should have a full set of tapes for each distinct operation in your backup schedule. For example, if you do a backup every day, it's best to have five sets of tapes that you reuse each week; if you can afford it, you might even have 20 sets that you rotate through every four weeks. Using a single set of tapes over and over again is inviting disaster.

Labeling tapes clearly is also a great help in finding the right one quickly later. Color-coded labels are favored by many sites as an easy yet effective way to distinguish the different sets of tapes. At the other extreme, I visited a site where the backup system they developed prints a detailed label for the tape at the conclusion of each backup.

Write-protect backup media. This prevents backup media from being accidentally overwritten. The mechanism for write-protection varies with different media types, but most mechanisms involve physically moving a plastic dial or tab to some designated position. The position that is the unwriteable one varies: floppy disks, optical disks and DAT (4mm) tapes are writeable when the tabbed opening is closed, while 8mm tapes and removable disks are writeable when it is open.

Consider the environment. Most backup media like it cool, dry, and dark. High humidity is probably the most damaging environment, especially for cartridge-enclosed media, which are easily ruined by the moisture condensation that accompanies temperature drops in humid conditions. Direct sunlight should also be avoided, especially for floppy disks, since most plastic materials will deform when subjected to the temperature within the trunk of a car or the enclosed passenger compartment on a hot summer day. Dust can also be a problem for most backup media. I've had lint make floppy disks unreadable after taking them home in my coat pocket (now I put them in a zip-top plastic bag first).

The fact that backup media prefer the same environment used for many computer rooms does not necessarily mean that any or all backup media should be stored in the same room as the computer. Doing so runs the risk that a major problem will destroy both the computer and the backups. Backup tapes are actually more sensitive to some types of problems than some computer components. For example, if a pipe bursts above the computer room, the computer may suffer only minor damage, but your backup tapes will usually all be ruined if they get wet.

If the tape storage area differs in temperature from the computer area by more than a few degrees, allow the tapes to acclimate to the computer temperature before writing to them.

Magnetic interference is also something to think about. One of this book's technical reviewers relayed a story about "an entire backup library that kept getting wiped out on a nearly daily basis. Turns out that the tapes were in a secure location but placed against a wall that was shared with a freight elevator. The magnetic fields and such caused by the moving lift caused all that nice magnetic tape storage to become erased. Funny but cautionary."

Handle media properly. Some media have special requirements that you'll need to take into account. For example, floppy disks and zip disks ideally should be stored upright, resting on a thin edge rather than stacked on top of one another. Similarly, cartridge tapes like to be stored with the spools vertical (perpendicular to the ground, like a car's tires) with the edge that contacts the drive heads down (so gravity pulls tape away from the spools). When you're counting on media to preserve important data, humor them and orient them the way they prefer.

Take security into account. In every location where you store backup tapes, the usual physical security considerations apply: the tapes should be protected from theft, vandalism, and environmental disasters as much as is possible.

Off-site and long-term storage

Off-site backups are the last barrier between your system and total annihilation. They are full backup sets that are kept in a locked, fireproof, environmentally-controlled location completely off site. Such backups should be performed on unmounted filesystems if at all possible.

Preparing backups for off-site storage is also one of the few times when simply making a backup is not enough.* In these cases, you also need to verify that the backup

* Another such time is when you are rebuilding a filesystem.

tape or diskette is readable. This is done by using an appropriate restore command to list the contents of the tape or diskette. While this will not guarantee that every file is completely readable, it will improve the odds of it considerably. Some backup utilities provide a full verification facility in which the entire content of each file in a backup set is compared with the corresponding file on disk; this is the preferred method of checking critical backups. In any case, backups should be verified in the best way available whenever the integrity of the backup is essential.

Permanent Backups

For data meant for permanent archiving, you should create and verify two sets of backup media with the idea that the redundant copy can be used should the first one fail. The media should also be checked periodically (annually or possibly biannually). When a particular media item fails—and they all will eventually—a new copy should be made from the other one to replace it.

You should also make sure that you have at least one working drive of the type that you are using for permanent storage media. For example, if you have an archive of 8 mm tapes, you will need to always have working 8 mm tape drives to read them. This will continue to be true if your primary backup medium changes. Similarly, you must maintain whatever software programs and other running environment is required to use the data for it to be of any use.

Finally, tapes should be rewound or retensioned regularly (perhaps twice a year) to maintain readability. Given this requirement, tapes are being superceded by CDs as permanent storage media.

When Being Compulsive is Good

It's very easy to put off doing backups, especially when you are responsible only for your own files. However, performing backups regularly is vital. Basically, it's a good idea to assume that the next time you sit down at the computer, all your disks will have had head crashes. Keeping such a catastrophe in mind will make it obvious what needs to be backed up and how often. Backups are convenient for restoring accidentally deleted files, but they are also essential in the event of serious hardware failures or other disasters. Catastrophes *will* happen. All hardware has a finite lifetime, and eventually something will fail.

Given this reality, it is obvious why an almost drone-like adherence to routine is an important attribute for an effective system administrator. Planning for worst-case scenarios is part of the job. Let them call you compulsive if they want to; one day, your compulsiveness—also known to many as carefulness—will save them, or at least their files.

Backup Media

When I first started working as a system administrator, 9-track tape was the only medium you'd consider using for a backup.[*] That's certainly no longer true. Today, there are many different media suitable for storing backed-up data. This section provides a quick summary of the available choices. This list includes most of the drives and media types which are in common use. The backup strategy for any particular system will often involve more than one media type.

Up-to-the-minute information about available backup devices and media may be obtained from *http://www.storagemountain.com*. There is also an excellent discussion in *Unix Backup and Recovery*.

Magnetic tape

Magnetic tape of one sort or another has been the traditional backup medium for decades. Over the years, it has taken on a variety of sizes and forms, beginning with 7-track and then 9-track tape: 1/2-inch wide tape wound around a circular reel. The introduction of plastic cartridges containing the tape and both reels was a major step forward in terms of reducing the space requirements of backup media. The first tape of this type was 1/4-inch cartridge tape (also known as QIC tape), which for a while was the medium of choice for most workstations; these tapes are still occasionally used.

Around 20 years ago, higher-capacity tapes in formats originally developed for other markets became available. 8 mm tape drives became popular in the late 1980s and are still in wide use. Originally designed for video uses, the tapes are about the size of an audio cassette. 4 mm digital audio tapes (commonly called DAT tapes although the data storage scheme is technically known as DDS) are also in wide use. DAT cartridges are about 25% smaller than 8 mm tapes.

8 mm and 4 mm tapes come in two grades, one designed for video and audio recording (respectively), and a better, more expensive grade designed for data. Be sure to purchase only data-quality tapes. Although lower-quality tapes will sometimes appear to work fine, in my experience they are much, much less reliable about retaining data (despite urban legends to the contrary).

Both of these tape types are in use today, although DAT is far more prevalent than 8 mm. Both types of tape come in a variety of lengths and corresponding data capacities. Currently, the largest ordinary 8 mm tapes are 160 meters long and hold up to

[*] The only other possibilities were punch cards and paper tape.

7 GB of data,[*] although there are also tapes that hold 1.2 GB (54 m) and 2.4 GB (112 m). DAT tapes correspond to various DDS levels:

DDS-1
: 2 and 3 GB tapes (60 and 90 m

DDS-2
: 4 GB tapes (120 m)

DDS-3
: 12 GB tapes (125 m)

DDS-4
: 20 GB tapes (150 m)

DDS-3 and DDS-4 use a different technology than the earlier versions.

Be aware that only the newest tape drives can support the largest tapes, but most drives provide read-only backward compatibility.

There are also several newer magnetic tape technologies. Exabyte's Mammoth-2[†] and Sony's Advanced Intelligent Tape (AIT) technologies take 8 mm tapes to much higher capacities: 20, 40, or 60 GB and 35 or 50 GB, respectively. They both use the Advanced Metal Evaporative (AME) cartridge developed by Sony (a new 8 mm format). Some Mammoth-2 drives can also read earlier 8 mm tapes, but they require an extensive clearing procedure to be performed after each instance. These are also among the fastest tape drives, with transfer rates of up to 12 MB/s for Mammoth-2 drives and 6 MB/s for AIT drives.

The Digital Linear Tape (DLT) technology was initially developed by Digital Equipment Corporation, but they later sold it to Quantum Corporation. This format uses cartridges similar to DEC's old TK family, which have proven themselves to be extremely reliable and long-lived. It is also a fast format, with transfer rates of up to 10 MB/s.

The high capacity of magnetic tapes make them ideal for unattended backups: you can put a tape in at night, start a shell script that puts several filesystems on one tape, and go home.

Tapes also have some disadvantages:

- They are extremely sensitive to heat and electromagnetic fields and fail quite easily when they are mishandled. Electromagnetic fields are produced by a variety of common devices found near computers, including UPS power supplies, external peripheral devices containing their own power supplies, monitors, and

[*] That is, 7 GB of bits. The amount of "data" written may be much more if the original files are compressed before or as they are written to tape. Tape drive and media manufacturers love to inflate their products' capacities by quoting maximum compressed data numbers.

[†] This was preceded by the Mammoth technology, which was notoriously unreliable. Mammoth-2 initially seems to be better.

speakers. Moreover, simply reading a magnetic tape also contributes to data degradation.

- They are sequential storage devices. In order to reach a given file on a tape, you have to wind the tape to the proper point. This is more of a problem for older tapes drives; current high-end drives can reach an arbitrary point on a tape in seconds.

Magneto-optical disks

Magneto-optical disks have the same width and length as floppy disks but are about twice as thick and hold a lot more data. Magneto-optical disks also come in 3.5-inch and 5.25-inch versions,[*] and their current capacity ranges up to 9.1 GB. Optical disks are purported to be much more stable than any of the purely magnetic media; the stability comes from the fact that they are written magnetically but are read optically, so reading the disk has no degrading effect on the stored data. In addition, the media can also be erased and rewritten as needed. Finally, magneto-optical disks also have the advantage of being random access devices. Transfer rates for these devices peak at about 5 MB/s.

Current drives are still quite expensive—over $2000—as are the disks themselves, but they are nevertheless very popular. As I noted in the previous edition of this book (circa 1995), "a rewriteable medium that can permanently store over a gigabyte of data in the space of a couple floppy disks probably has a future." Now it's gigabytes of data and a definite future.

> There are also other optical formats used or in development by a few manufacturers.

CDs and DVDs

Writeable CDs and DVDs have become viable backup media due to the substantial price reductions for both drives and media. There are two types of writeable CDs, referred to as CD-R (write-once CDs) and CD-RW (rewriteable CDs). Both come in 640 MB capacity, and recently 700 MB CD-R media have become available.

Writeable DVD technology is just emerging into the general marketplace at this writing. In fact, there are several DVD recording formats:

DVD-RAM
: The first available format, it is now falling out of use since it cannot be read in ordinary DVD drives.

[*] You might wonder what is so magical about 3.5 and 5.25 inches. Devices of this size fit easily into the standard device bays found in PCs as well as into available storage boxes.

DVD-R
: Write-once DVDs (also an aging technology).

DVD-RW
: Rewriteable DVDs that can be read by ordinary DVD drives.

DVD+RW
: An emerging technology devised by a coalition of drive manufacturers. These drives can produce ordinary (sequential) DVDs as well as random access disks. The former are readable by ordinary DVD players (but not by recorders of the other types), although some older models may require firmware updates. DVD+RW media can hold up to 4.7 GB per side.

: As of this writing, Hewlett-Packard has recently released a low-cost DVD+RW writer suitable for use on PC-based systems, so this may become a popular backup device in that market in time.

Removable disks: Zip and Jaz

Removable disks are fully enclosed disk units that are inserted into a drive as needed. They tend to be significantly more reliable than either tapes or floppy disks. On Unix systems, they generally behave like a hard disk, but it is also possible to treat them as a giant floppy disk. They are suitable as backup media in some environments and circumstances.

There have been a variety of removable disk technologies over the years. The Zip and Jaz drives by Iomega have come to dominate this market. Zip drives—which come in 100 MB and 250 MB sizes—can be used with most Unix systems. Jaz drives, which have capacities of 1 GB or 2 GB, can also be used. I had a great deal of trouble with early Jaz drives, which were designed for infrequent, intermittent backup use and consistently failed when used on even a semi-continuous basis. More recent drives are said to be better. Both drive types are available with various I/O interfaces: SCSI, USB, IDE.

Floppy disks

Floppy disk drives are still found on most PC-based computer systems,* and they do have some limited backup uses. For example, PC-based Unix versions (as well as a few running on larger systems) often use floppy disks for emergency boot devices. In addition, floppy disks can be useful for inherently limited backup tasks, such as saving customized system configuration files from the root filesystem. Standard floppy disks hold 1.44 MB, and some Unix workstations include drives that double that capacity to 2.8 MB. Occasionally, you will come across a floppy drive that also supports Super disks: media that look like floppy disks but hold 120 MB.

* Although this will probably no longer be true in a couple of years.

Hard disks

Given the low prices of hard disks these days, they may also be a viable backup target device in some circumstances. For example, some sites provide a large backup disk on the local network where users can make periodic copies of key files that they are working on. Large disks can also be used for scratch purposes, for temporary data repositories and data holding areas, and similar purposes. They can also be used as a staging area where backups are stored temporarily on the way to being written to tape or other media.

Stackers, jukeboxes, and similar devices

There are a variety of devices designed to make media handling more automated, as well as to store and make available large numbers of media units. For example, there are auto-loading tape drives—also known as *stackers*—which can feed tapes automatically from a stack of 10 or so. Early stackers could access the component tapes only in order, but many current devices can retrieve any desired tape.

Another type of device puts multiple drive units into a box that looks to users like a single tape drive with the combined capacity of all of its components. Alternatively, such a device can be used to make multiple identical tapes simultaneously.

Still other units combine both multiple drives and tape auto-loading capabilities. These devices are known as *jukeboxes* or *libraries*.* The most sophisticated of them can retrieve a specific tape and place it into the desired drive. Some of these devices include integrated bar-code readers so that tapes can be identified by their physical label rather than storage location or electronic label. Similar devices also exist for optical disks and writeable CD-ROMs.

Media Lifetime

From time to time, you also need to think about the reasonable expected lifetime of your backup media. Stored under the right conditions, tapes can last for years, but unfortunately you cannot count on this. Some manufacturers recommend replacing tapes every year. This is certainly a good idea if you can afford to do so. The way that tapes and diskettes are stored also affects their lifetime: sunlight, heat, and humidity can all significantly shorten it. I always replace tapes that have had read errors or other failures more than once, regardless of their age; for some people and situations, a single failure is enough. I always throw away diskettes and Zip disks at the first hint of trouble.

* Very large libraries (greater than 500 volumes) are known as *silos*. The two types of devices used to be distinguished by whether or not multiple hosts could be connected, but some libraries now have this capability. Separate silos are also able to pass tapes between them.

Even so-called permanent media like CDs actually have a finite lifetime. For example, CDs begin to fail after about 5 years (and sometimes even sooner). Accordingly, creating two copies of important data and checking them periodically is the only prudent course.

Given these considerations, your site may want to consider alternative media for off-site and archival backups. For example, manufacturers of optical disks claim a lifetime of 15 years for this media (this is based on accelerated aging tests; as of this writing, we won't know for about 8–9 years whether this is really true).

Comparing Backup Media

Table 11-1 lists the most important characteristics of a variety of backup media. The largest media capacity for each item shown is the biggest that was available as of this writing. These size values refer to raw data capacity: the actual amount of data that can be written to the media.

The drive price is the lowest generally available price at this time and can be assumed to use the least expensive I/O interface; you can expect SCSI versions of many devices that are also available in IDE form to cost at least 15% more (and sometimes much more). Similarly, at about $100, a USB floppy drive costs 10 times that of an ordinary one.

The media prices are the lowest commonly available when the media is purchased in large quantities (e.g., 50–100 for CDs) and in no-frills packaging (e.g., on a spindle for CDs rather than in individual jewel cases). All prices are domestic prices in the United States, in U.S. dollars, as of mid-2002.

The minimum lifetime column gives an approximate rule-of-thumb time period when you can expect some media to begin failing. Of course, individual media will fail even sooner in some cases.

Table 11-1. Popular backup devices and media

Type	Media capacity	Drive price[a]	Media price[a]	Minimum lifetime
Floppy disk	1.44 MB[b]	$10	$0.25	2 years
Super disk	120 MB	$120	$8	2–3 years
Zip Disk	100 MB	$70	$5	3–5 years
	250 MB	$140	$12	3–5 years
Jaz Disk	1 GB	$300	$80	4–5 years
	2 GB	$340	$100	4–5 years
CD-R	700 MB (80 minutes)	$150	$0.85	5 years
CD-RW	640 MB (74 minutes)	$150	$1	5 years
DVD-R	4.7 GB (single-sided)	$700	$8	5 years?
	9.4 GB (double-sided)	$700	$40	5 years?

Table 11-1. Popular backup devices and media (continued)

Type	Media capacity	Drive price[a]	Media price[a]	Minimum lifetime
DVD+RW	4.7 GB	$600	$8	5 years?
DAT tape 4 mm DDS	4 GB (120 m DDS-2)	$550	$6	3–4 years
	12 GB (125 m DDS-3)	$700	$12.50	3–4 years
	20 GB (150 m DDS-4)	$1200	$26	3–4 years
8 mm tape	7 GB (160 m)	$1200	$6	2–4 years
Mammoth-2 (AME)	20 GB	$2500	$36	3–4 years?
	60 GB	$3700	$45	3–4 years?
AIT tape	35 GB	$900	$79	3–4 years?
	50 GB	$2600	$85	3–4 years?
	100 GB	$3900	$105	3–4 years?
DLT	40 GB	$4000	$70	10 years
SuperDLT	110 GB	$6000	$150	10 years
Magneto-optical (RW)	5.2 GB	$2300	$65	15 years?
	9.1 GB	$2700	$93	15 years?
Hard disk	100 GB (IDE)	N/A	$2–3/GB	5–7 years
	180 GB (SCSI)	N/A	$10/GB	5–7 years

[a] Approximate minimum price in U.S. dollars.
[b] A few floppy drives provided by Unix vendors increase the maximum capacity to 2.8 MB.

Tape Special Files

Traditionally, special files used to access tape drives had names of the form */dev/rmt*n or */dev/rmt/*n, where *n* indicates the drive number. Tape drives are virtually always accessed via the character (raw) special file. Currently, special file names usually include other characters as prefixes and/or suffixes, which indicate the way the device is to be accessed: the density setting to use, whether to use the drive's built-in hardware compression, whether to rewind the tape after the operation is completed, and so on.

AIX systems also use suffixes to select whether the tape should be retensioned before use. Retensioning refers to equalizing the tension on a tape, and it consists of moving the tape to the beginning, then the end, and then rewinding back to the beginning; it's even slower than it sounds. The idea is to eliminate any latent slackness in the tape, but it is seldom necessary in practice.

Table 11-2 lists the current tape special file naming conventions for the various operating systems we are considering.

Table 11-2. Tape special file names

Unix version	Format and examples[a]	Prefixes/suffixes	man page
AIX	*/dev/rmt*n[.m] */dev/rmt0.1* */dev/rmt0.5* **Note**: Compression is enabled and disabled with the `chdev` command.	m: none=rewind, no retension, low density *1*=no rewind, no retension, low density *2*=rewind, retension, low density *3*=no rewind, retension, low density *4*=rewind, no retension, high density *5*=no rewind, no retension, high density *6*=rewind, retension, high density *7*=no rewind, retension, high density	`rmt(4)`

FreeBSD	*/dev/*[*n*]*rast*n
	/dev/[*e*|*n*]*rsa*n
	/dev/nrast0
	/dev/nrsa0
		n=no rewind
		e=eject tape on close
		(Density and compression are chosen with the `mt` utility.)
			`ast`
			`sa(4)`

HP-UX	*/dev/rmt/c*i*t*j*d0*TYPE[*b*][*n*] */dev/c0t3d0DDSbn* */dev/c0t3d0BESTbn*	i=controller j=SCSI ID *n*=no rewind *b*=use BSD-style error control TYPE=keyword indicating tape type and/or density (e.g., *BEST*, *DDS*)	`mt(7)`
Linux	*/dev/*[*n*]*st*nx */dev/nst0* */dev/nst0m*	*n*=no rewind x: none=default density *l*=low density *m*=medium density *a*=autoselect density	`st`
Solaris	*/dev/rmt/*nx[*b*][*n*] */dev/rmt/0lbn* */dev/rmt/0hbn*	*b*=use BSD-style error control *n*=no rewind x: none=default density *l*=low density *m*=medium density *h*=high density *c*=use hardware compression	`st`
Tru64[b]	*/dev/*[*n*]*rmt/tape*n_dm */dev/nrmt/tape0_d2* */dev/nrmt/tape0_d3*	m: *0*=low density, use compression *1*=high density, use compression *2*=low density, no compression *3*=high density, no compression (values 4–7 are also defined for some drives)	`tz`

[a] In all cases, n refers to the tape drive number. The examples are all for a non-rewinding tape device with hardware compression disabled using the lowest and highest density (as available).

[b] Older Tru64 systems use the now-obsolete device names of the form */dev/tz** and */dev/ta**.

Some systems provide simpler names as links to commonly-used tape devices. You can figure out which device they refer to by looking at a long directory listing. Here is an example from an HP-UX system:

```
crw-rw-rw- 2 bin bin 205 0x003000 Oct 7 1999 0m
crw-rw-rw-  2 bin    bin    205 0x003080 Oct  7  1999 0mb
crw-rw-rw-  2 bin    bin    205 0x003040 Oct  7  1999 0mn
crw-rw-rw-  2 bin    bin    205 0x0030c0 Oct  7  1999 0mnb
crw-rw-rw-  2 bin    bin    205 0x003000 Oct  7  1999 c0t3d0BEST
crw-rw-rw-  2 bin    bin    205 0x003080 Oct  7  1999 c0t3d0BESTb
crw-rw-rw-  2 bin    bin    205 0x003040 Oct  7  1999 c0t3d0BESTn
crw-rw-rw-  2 bin    bin    205 0x0030c0 Oct  7  1999 c0t3d0BESTnb
crw-rw-rw-  1 bin    bin    205 0x003001 Oct  7  1999 c0t3d0DDS
crw-rw-rw-  1 bin    bin    205 0x003081 Oct  7  1999 c0t3d0DDSb
crw-rw-rw-  1 bin    bin    205 0x003041 Oct  7  1999 c0t3d0DDSn
crw-rw-rw-  1 bin    bin    205 0x0030c1 Oct  7  1999 c0t3d0DDSnb
```

In this case, *0m* and *c0t3d0BEST* refer to the same tape drive and access mode (as do their corresponding suffixed forms).

The default tape drive on a system is usually the first drive in its default (rewinding) mode:

AIX	*/dev/rmt0*
FreeBSD	*/dev/rsa0*
HP-UX	*/dev/rmt/0m*
Linux	*/dev/st0*
Solaris	*/dev/rmt/0*
Tru64	*/dev/rmt/tape0_d0*

On Linux systems (and some others), the device */dev/tape* is a link to the default tape device on the system. You can make the link point to whatever drive you want to by recreating the link. On FreeBSD systems, some commands use the *TAPE* environment variable to locate the default tape drive.

AIX tape device attributes

On AIX systems, you can use the lsattr command to view the attributes of a tape drive:

```
$ lsattr -E -H -l rmt0
attribute      value  description                      user_settable

block_size     1024   BLOCK size (0=variable length)     True
compress       yes    Use data COMPRESSION               True
density_set_1  140    DENSITY setting #1                 True
density_set_2  20     DENSITY setting #2                 True
extfm          yes    Use EXTENDED file marks            True
mode           yes    Use DEVICE BUFFERS during writes   True
```

This 8 mm tape drive will use data compression and a block size of 1024 by default.

You must use the chdev command to change the many attributes of a tape drive (rather than having these selections encoded into the special file name as with other systems). For example, the following command changes the block size to 1024 and turns off compression and retensioning for drive 1:

```
# chdev -l rmt0 -a block_size=1024 -a compress=no -a ret=no
```

Backing Up Files and Filesystems

Most systems offer a variety of utilities for performing backups, ranging from general-purpose archiving programs like tar and cpio to programs designed for implementing multilevel incremental backup schemes on a per-filesystem basis. When the largest tapes held only a couple hundred megabytes, choosing the right utility for system backups was easy. tar and cpio were used for small and ad hoc backups and other data transfer needs, and the more sophisticated utilities specifically designed for the task were used for system backups, because their specialized abilities—the ability to span tapes and to automatically perform incremental backups—were essential to getting the job done.

This distinction breaks down to a great extent when a single tape can hold gigabytes of data. For example, incrementals are less important when you can fit all the important data on a system onto one or two tapes—and you have the time to do so. Large tapes also make it practical to back up a system in logically grouped chunks of files, which may be spread arbitrarily throughout the physical filesystem. A successful system backup process can be built around whatever utilities make sense for your system.

One dubious piece of advice about backups that is frequently given is that you should limit filesystem size to the maximum backup media capacity available on the system. In this view, multi-tape backup sets are simply too much trouble, and the backup process is simplified if all of the data from a filesystem will fit onto a single tape.

While being able to back up a filesystem with a single tape is certainly convenient, I think it is a mistake to let current media capacity dictate filesystem planning to such a degree. Breaking disks into more, smaller filesystems limits flexibility in allocating their resources, a concern that is almost always far more important than reducing the complexity of backing them up. Designing the filesystem needs to take *all* of the factors affecting the system and its efficient use into account. If tape-sized backup sets are what is desired, it's easy enough to write scripts to do so when overall circumstances dictate that some individual filesystems need to be bigger.

When tar or cpio Is Enough

In some cases, especially single-user systems, an elaborate backup process is not needed. Rather, since the administrator and the user are one and the same person, it will be obvious which files are important, how often they change, and so on. In cases like this, the simpler tape commands, `tar` and `cpio`, may be sufficient to periodically save important files to tape (or other media).

While the canonical model for this situation is Unix running on a workstation, these utilities may also be sufficient for systems with relatively small amounts of critical data. `tar` and `cpio` also have the advantage that they will back up both local and remote filesystems mounted via NFS.

The tar command

We'll begin with a simple example. The following `tar` command saves all files under */home* to the default tape drive:

```
$ tar -c /home
```

`-c` says to create a backup archive.

`tar`'s `-C` option (big C) is useful for gathering files from various parts of the filesystem into a single archive. This option causes the current directory to be set to the location specified as its argument before `tar` processes any subsequent pathname arguments. Multiple `-C` options may be used on the same command. For example, the following `tar` commands save all the files under the directories */home*, */home2*, and */chem/public*:

```
$ tar -cf /dev/rmt1 /home /home2 /chem/public
$ tar -cf /dev/rmt1 -C /home . -C /home2 . -C /chem public
```

The two commands differ in this respect: the first command saves all of the files using absolute pathnames: */home/chavez/.login*, for example. The second command saves files using relative pathnames: *./chavez/.login*. The file from the first archive would always be restored to the same filesystem location, while the file from the second archive would be restored relative to the current directory (in other words, relative to the directory from which the restore command was given).

It is a good idea to use absolute pathnames in the arguments to `-C`. Relative pathnames specified to `-C` are interpreted with respect to the current directory at the time that option is processed rather than with respect to the initial current directory from which the `tar` command was issued. In other words, successive `-C` options accumulate, and `tar` commands using several of them as well as relative pathnames can become virtually uninterpretable.

Traditionally, all tar options were placed in a single group immediately following the command verb, and a preceding hyphen was not needed. The POSIX standard specifies a more traditional Unix syntax, preferring the second form to the first one for this command:

```
$ tar xpfb /dev/rmt1 1024 ...
$ tar -x -p -f /dev/rmt1 -b 1024 ...
```

The versions of tar on current operating systems usually accept both formats, but an initial hyphen may become be a requirement at some point in the future.

tar archives are often compressed, so it is very common to see compressed tar archives with names like *file.tar.Z*, *file.tar.gz* or *file.tgz* (the latter two files are compressed with the GNU gzip utility).

Solaris enhancements to the tar command. The Solaris version of tar offers enhancements that make the command more suitable for system-level backups. They allow all or part of the list of files and directories to be backed up to be placed in one or more text files (with one item per line). These files are included in the file list given to tar, preceded by -I, as in this example:

```
$  tar cvfX /dev/rst0 Dont_Save /home -I Other_User_Files -I Misc
```

This command backs up the files and directories in the two include files, as well as those in */home*. The command also illustrates the use of the -X option, which specifies the name of an exclusion file listing the names of files and directories that should be skipped if encountered by tar. Note that wildcards are not permitted in either include or exclusion files. In case of conflicts, exclusion takes precedence over inclusion.

The -I and -X options may also be used in restore operations performed with the tar command.

On Solaris and a variety of other System V systems, the file */etc/default/tar* may be used to customize the mappings of the default archive destinations specified with tar's single-digit code characters (for example, the command tar 1c creates an archive on drive 1). Here is a version from a Solaris system:

```
#                                Block   #
#Archive=Device                  Size    Blocks
#
archive0=/dev/rmt/0              20      0
archive1=/dev/rmt/0n             20      0
archive2=/dev/rmt/1              20      0
archive3=/dev/rmt/1n             20      0
archive4=/dev/rmt/0              126     0
archive5=/dev/rmt/0n             126     0
```

```
archive6=/dev/rmt/1            126     0
archive7=/dev/rmt/1n           126     0
```

The first entry specifies the device that will be used when `tar 0` is specified. In this case, it is the first tape drive in its default modes. The second entry defines archive 1 as the first tape drive in non-rewinding mode. The remaining two fields are optional; they specify the block size for the device and its total capacity (which may be set to zero to have the command simply detect the end-of-media marker).

The GNU tar utility: Linux and FreeBSD. Linux distributions and FreeBSD provide the GNU version of the `tar` command. It supports `tar`'s customary features and contains some enhancements to them, including the ability to optionally span media volumes (`-M`) and to use `gzip` compression (`-z`). For example, the following command will extract the contents of the specified compressed `tar` archive:

```
$ tar xfz funsoftware.tgz
```

The cpio command

`cpio` can also be used to make backups. It has several advantages:

- It is designed to easily back up completely arbitrary sets of files; `tar` is easiest to use with directory subtrees.
- It packs data on tape much more efficiently than `tar`. If fitting all your data on one tape is an issue, `cpio` may be preferable.
- On restores, it skips over bad spots on the tape, while `tar` just dies.
- It can span tapes, while many versions of `tar` are limited to a single volume.

Using its `-o` option, `cpio` copies the files whose pathnames are passed to it via standard input (often by `ls` or `find`) to standard output; you redirect standard output to use `cpio` to write to floppy disk or tape. The following examples illustrate some typical backup uses of `cpio`:

```
$ find /home -print | cpio -o >/dev/rmt0
$ find /home -cpio /dev/rmt0
```

The first command copies all files in */home* and its subdirectories to the tape in drive 0. The second command performs the identical backup via a version of `find` that offers a `-cpio` option.

Incremental backups with tar and cpio

Combining `find` with `tar` or `cpio` is one easy way to perform incremental backups, especially when only two or three distinct backup levels are needed. For example, the following commands both copy all files under */home* which have been modified today into an archive on */dev/rmt1*, excluding any object (*.o*) files:

```
$ find /home -mtime -1 ! -name \*.o -print | cpio -o >/dev/rmt1
$ tar c1 `find /home -mtime -1 ! -name `*.o' ! -type d -print`
```

The find command used with tar needs to exclude directories, because tar will automatically archive *every* file underneath any directory named in the file list, and all directories in which *any* file has changed will appear in the output from find.

You can also use find's -newer option to perform an incremental backup in this way:

```
$ touch /backup/home_full
$ find /home -print | cpio -o > /dev/rmt0
A day later...
$ touch /backup/home_incr_1
$ find /home -newer /backup/home_full -print | cpio -o > /dev/rmt0
```

The first command timestamps the file */backup/home_full* using the touch command (*/backup* is a directory created for such backup time records), and the second command performs a full backup of */home*. Some time later, the second two commands could be used to archive all files that whose data has changed since the first backup and to record when it began. Timestamping the record files before this backup begins ensures that any files that are modified while it is being written will be backed up during a subsequent incremental, regardless of whether such files have been included in the current backup or not.

pax: Detente between tar and cpio

The pax command attempts to bridge the gap between tar and cpio by providing a single general-purpose archiving utility.[*] It can read and write archives in either format (by default, it writes tar archives), and offers enhancements over both of them, making it an excellent utility for system backups in many environments. pax is available for all of the Unix versions we are considering. Like cpio, pax archives may span multiple media volumes.

pax's general syntax is:

```
pax [mode_option] other_options files_to_backup
```

The *mode_option* indicates whether files are being written to or extracted from an archive, where -w says to write to an archive, -r says to read and extract from an archive, and -rw indicates a pass-through mode in which files are copied to an alternate directory on disk (as with cpio -p); pax's default mode when no *mode_option* is given is to list the contents of an archive.

The following commands illustrate pax file archiving modes of operation:

```
$ pax -w -f /dev/rmt0 /home /chem
$ find /home /chem -mtime -1 -print | pax -w -f /dev/rmt0
$ pax -w -X -f /dev/rmt0 /
```

[*] Indeed, on systems offering pax, cpio and tar are often just links to it. pax's syntax is an amalgamation of the two, which is not surprising for a peace imposed by POSIX (although the name purportedly stands for *portable archive exchange*).

The first two commands perform a full and incremental backup of the files in */home* and */chem* to the default tape drive in each case. The third command saves all of the files in the disk partition corresponding to the root directory; the `-X` option tells `pax` not to cross filesystem boundaries.

AIX prefers `pax` over vanilla `tar` and `cpio`. The command has been enhanced to support large files (over 2 GB).

Getting Users to Do Backups

At some sites, certain backup responsibilities are left to individual users: when a site has far too many workstations to make backing up all of their local disks practical, when important data resides on non-Unix systems like PCs (especially if they are not connected to the local area network), and so on.

However, even when you're not actually performing the backups yourself, you will probably still be responsible for providing technical support and, more often than not, reminders to the users who will be performing the backups. Here are some approaches I've tried to facilitate this:

- Make a habit of encouraging users rather than threatening them (threats don't work anyway).
- Use peer pressure to your advantage. Setting up a central backup storage location that you look after can make it obvious who is and isn't doing the backups they are supposed to. Note that this idea is inappropriate if data sensitivity is an issue.
- Create tools that automate the backup process as much as possible for users. Everyone has time to drop in a tape and start a script before they leave for the day.
- Provide a central repository for key files that get backed up as part of the system/site procedure. Users can copy key files and know they will be backed up when they're really in a jam and really don't have time to do a backup themselves.

Backing Up Individual Filesystems with dump

The BSD `dump` utility represents the next level of sophistication for backup systems under Unix. It selectively backs up all of the files within a filesystem (single disk partition), doing so by copying the data corresponding to each inode to the archive on the backup device. It also has the advantage of being able to back up any type of file, including device special files and sparse files. Although there are slight variations among different versions of this command, the discussion here applies to the following Unix implementations of this command:

AIX	`backup`
FreeBSD	`dump`

HP-UX	`dump` and `vxdump`
Linux	`dump` (but the package is not usually installed by default)
Solaris	`ufsdump`
Tru64	`dump` and `vdump`

On systems supporting multiple filesystem types, `dump` may be limited to UFS (BSD-type) filesystems; on Linux systems, it is currently limited to ext2/ext3 filesystems, although the XFS filesystem provides the similar `xfsdump` utility. Under HP-UX, `vxdump` and `vxrestore` support VxFS filesystems. Tru64 provides `vdump` for AdvFS filesystems.

`dump` keeps track of when it last backed up each filesystem and the level at which it was saved. This information is stored in the file */etc/dumpdates* (except on HP-UX systems, which use */var/adm/dumpdates*). A typical entry in this file looks like this:

```
/dev/disk2e       2 Sun Feb  5 13:14:56 1995
```

This entry indicates that the filesystem */dev/disk2e* was last backed up on Sunday, February 5 during a level 2 backup. If `dump` does not find a filesystem in this list, it assumes that it has never been backed up.

If *dumpdates* doesn't exist, the following command will create it:

```
# touch /path/dumpdates
```

The *dumpdates* file must be owned by the user *root*. If it does not exist, `dump` will not create it and won't record when filesystem backups occur, so create the file before running `dump` for the first time.

The `dump` command takes two general forms:

```
$ dump options-with-arguments filesystem
$ dump option-letters corresponding-arguments filesystem
```

where *filesystem* is the block special file corresponding to the filesystem to be backed up or the corresponding mount point from the filesystem configuration file. In the first, newer form, the first item is the list of options to be used for this backup, with their arguments immediately following the option letters in the normal way (e.g., `-f /dev/tape`).

In the second, older form, *option-letters* is a list of argument letters corresponding to the desired options, and *corresponding-arguments* are the values associated with each argument, in the same order. This syntax is still the only one available under Solaris and HP-UX.

Although not all options require arguments, the list of arguments must correspond *exactly*, in order and in number, to the options requiring arguments. For example, consider the set of options `0sd`. The `s` and `d` options require arguments; `0` does not. Thus, a `dump` command specifying these options must have the form:

```
$ dump 0sd s-argument d-argument filesystem
```

Failing to observe this rule can have painful consequences if you are running the command as *root*, including destroying the filesystem if you swap the argument to the `f` option and `dump`'s final argument when you are running the command as *root*. You'll get no argument from me if you want to assert that this is a design defect that ought to have been fixed long before now. When you use `dump`, just make sure an argument is supplied for each option requiring one. To avoid operator errors, you may want to create shell scripts that automatically invoke `dump` with the proper options.

`dump`'s most important options are the following (we will use the newer form):

`-0`, ..., `-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 filesystem. If there is no record of a backup for this filesystem 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.

: Older versions of `dump` not supporting hyphenated options require that the level option be the first option letter.

`-u`
: If `dump` finishes successfully, this option updates its history file, *dumpdates*. It does not require an argument.

`-f` *`device`*
: 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). The defaults used by various Unix versions were listed previously. 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.

`-W`
: Display only what will be backed up when the indicated command is invoked, but don't perform the actual backup operation.

`-s` *`feet`* `-d` *`dens`*
: These options were needed on older versions of `dump` to determine the capacity of the backup media. Recent versions of `dump` generally don't need them as they keep writing until they detect an end-of-media mark.

: If you do need to use them to lie to `dump` about the tape length because your version uses a default capacity limit suitable for ancient 9-track tapes, `-s` specifies the *size* of the backup tape, in feet; `-d` specifies the density of the backup tape, in bits per inch. Since `dump` will respect end-of-media marks that it encounters before it has reached this limit, the fix for such situations is to set the capacity to something far above the actual limit. For example, the options `-d 50000 -s 90000` define a tape capacity somewhat over 4 GB.

`-b` *`factor`*
: Specifies the block size to use on the tape, in units of 1024-byte (or sometimes 512-byte) blocks.

Here is a typical use of the `dump` command:

```
$ dump -1 -u -f /dev/tape /chem
```

The second command performs a level 1 incremental backup on the */chem* filesystem using the tape drive linked to */dev/tape*; `dump` will update the file the *dumpdates* file upon completion.

`dump` notifies the user whenever it requires some interaction. Most often, `dump` will have filled the tape currently in use and ask for another. It will also ask whether to take corrective actions if problems arise. In addition, `dump` prints many messages describing what it is doing, how many tapes it thinks it will need, and the like.

The HP-UX fbackup utility

HP-UX provides the `fbackup` and `frecover` utilities designed to perform system backups. One significant advantage that they have over the standard Unix utilities is that they can save and restore HP-UX access control lists along with other file metadata.

`fbackup` provides for nine levels of incremental backups, just like `dump`. `fbackup` stores backup records in the file */var/adm/fbackupfiles/dates*, which the system administrator must create before using `fbackup`.

The following examples illustrate how `fbackup` might be used for system backup operations:

```
# fbackup -0u -f /dev/rmt/1m -i /chem
# fbackup -1u -i /chem -i /bio -e /bio/med
# fbackup -1u -f /dev/rmt/0m -f /dev/rmt/1m -i /chem
# fbackup -0u -g /backup/chemists.graph -I /backup/chemists.TOC
```

The first command performs a full backup of */chem* to tape drive 1, updating the `fbackup` database. The second command does a level 1 backup of */chem* and */bio*, excluding the directory */bio/med* (as many `-i` and `-e` options as you need can be included). The third command performs a level 1 backup of */chem* using multiple tape drives in sequence.

The final command performs a full backup as specified by the graph file */backup/chemists.graph*, writing an index of the backup to the file */backup/chemists.TOC*. A graph file is a text file with the following format:

```
c    path
```

where *c* is a code indicating whether *path* is to be included (`i`) or excluded (`e`) from the backup.

Related Tape Utilities

There are two other Unix tape utilities you should know about, which are also of use in performing backups from time to time.

Data copying and conversion with dd

The `dd` utility transfers raw data between devices. It is useful for converting data between systems and for reading and writing tapes from and to non-Unix systems. It takes a number of *`option=value`* pairs as its arguments. Some of the most useful options are:

`if`
: Input file: source for data.

`of`
: Output file: destination for data.

`ibs`
: Input block size, in bytes (the default is 512).

`obs`
: Output block size, in bytes (the default is 512).

`fskip`
: Skip tape files before transferring data (not available in all implementations).

`count`
: The amount of data (number of blocks) to transfer.

`conv`
: Keyword(s) specifying desired conversion of input data before outputting: `swab` means swap bytes, and it is the most used conversion type. `lcase` and `ucase` mean convert to lower- and uppercase, respectively, and `ascii` and `ebcdic` mean convert to ASCII or EBCDIC.

For example, the following command processes the third file on the tape in drive 0, using an input block size of 1024 bytes and swapping bytes in all data; the command writes the converted output to the file */chem/data/c70o.dat*:

```
$ dd if=/dev/rmt0 of=/chem/data/c70o.dat \
     ibs=1024 fskip=2 conv=swab
```

As always, be careful to specify the appropriate devices for `if` and `of`; transposing them can have disastrous consequences.

Tape manipulation with mt

Unix provides the `mt` command for direct manipulation of tapes. It can be used to position tapes (to skip past backup save sets, for example), to rewind tapes, and to perform other basic tape operations. Its syntax is:

```
$ mt [-f tape-device] command
```

where *tape-device* specifies which tape drive to use, and *command* is a keyword indicating the desired action. Useful keywords include `rewind` (to rewind the tape), `status` (display device status—you can see whether it is in use, for example), `fsf` *n* (skip the next *n* files), and `bsf` *n* (skip back *n* files).

For example, to rewind the tape in the second tape drive, you might use a command like:

```
$ mt -f /dev/rmt1 rewind
```

The Solaris version of `mt` includes an `asf` subcommand, which moves the tape to the *n*th file on the tape (where *n* is given as `asf`'s argument), regardless of the tape's current position.

Under FreeBSD, the `mt` command is used to set the tape drive density and compression:

```
$ mt -f /dev/nrsa0 comp on density 0x26
```

AIX also includes the `tctl` utility (to which `mt` is really a link). `tctl` has the same syntax as `mt` and offers a few additional seldom-wanted subcommands.

Restoring Files from Backups

All of the backup facilities described in the previous sections have corresponding file restoration facilities. We'll look at each of them in turn in this section.

Restores from tar and cpio Archives

Individual files or entire subtrees can be restored easily from `tar` and `cpio` archives. For example, the following pairs of commands restore the file */home/chavez/freeway/quake95.data* and user *harvey*'s home directory (respectively) from an archive made of */home* located on the tape in the default tape drive (here, we use */dev/rmt0* for as the example location):

```
$ tar -xp /home/chavez/freeway/quake95.data
$ cpio -im '*quake95.data' < /dev/rmt0
$ tar -xp /home/harvey
$ cpio -imd '/home/harvey*' < /dev/rmt0
```

The `-p` option to `tar` and `-m` option to `cpio` ensure that all file attributes are restored along with the file. `cpio`'s `-d` option creates subdirectories as necessary when restoring a directory subtree (`tar` does so by default).[*]

[*] The second `cpio` command also assumes that there is no file or directory in */home* that begins with "harvey" other than user *harvey*'s home directory.

Restores with pax are similar. For example, the first of the following commands lists the files on the tape in drive 0, and the remaining commands extract various files from it:

```
$ pax -f /dev/rmt0 -v                    –v gives a more detailed/verbose listing.
$ pax -r '/h95/*.exe'                    Select files via a regular expression.
$ pax -r /home/chavez                    Restore chavez's home directory.
$ pax -r -f my_archive -c '*.o'          Restore everything except object files.
# pax -r -pe -f /dev/rmt0                Restore files incl. owner, mode & mod. time.
```

pax's coolest feature has to be its -s option, which allows you to massage filenames as files are written to, extracted from, or even just listed from an archive. It takes a substitution command as used in ed or sed as its argument (which will usually need to be enclosed in single quotation marks) indicating how filenames should be transformed. For example, the following command changes the second-level directory name of each file from *chavez* to *harvey* as files are read from the archive, changing their target location on disk:

```
$ pax -r -s ',^/home/chavez/,/home/harvey/,' \
      -f /dev/rmt0 /home/chavez
```

The substitution clause searches for */home/chavez* at the beginning of the pathname of each file to be restored and changes it to */home/harvey*, using commas as the field separator within the substitution string.

Here are some additional -s clauses for specific kinds of transformations:

```
-s ',^/home/chavez/,,'      Remove partial directory component.
-s ',^.*//*,,'              Remove entire directory component.
-s ',^//*,,'                Make pathnames relative to current directory.
```

Multiple -s options are allowed, but only the first matching one is used for any given filename.

Be aware that pax is not without its eccentricities. One of the most annoying is the following: in some versions of pax, directories matched via wildcards in the pattern list during restore operations are not extracted in their entirety; only explicitly listed ones are. Note that this is the opposite of the way cpio works and also counter to the way tar operates. I'd be positive this was a bug except that it happens in more than one vendor's version, although not in every vendor's version. With pax, *caveat emptor* would appear to be the watchword.

Restoring from dump Archives

The restore utility retrieves files from backup tapes made with the dump utility. It is supported by those systems supporting a version of dump. Solaris calls its version ufsrestore in keeping with the name of its version of dump. HP-UX and Tru64 provide vxrestore and vrestore commands for their default filesystem types. All of these commands have the same syntax and options. The commands can restore single files, directories, or entire filesystems.

To restore an entire filesystem, you must restore the most recent backup tapes from *each* backup level: the most recent full dump (0), the most recent level 1 dump, and so on. You must restore each level in numerical order, beginning with level 0. `restore` places the files it retrieves in the current working directory. Therefore, to restore a filesystem as a whole, you may wish to create and mount a clean, empty filesystem, make the current working directory the directory in which this filesystem is mounted, and then use `restore` to read the backup tapes into this directory. Note that such `restore` operations will have the side effect of recreating deleted files.

After a full restore, you need to do a full (level 0) backup. The reason for this is that `dump` backs up files by their inode number internally, so the tape from which you just restored from won't match the inodes in the new filesystem since they were assigned sequentially as files were restored.

In general, the `restore` command has the following forms (similar to `dump`'s):

```
$ restore options-with-arguments [files-and-directories]
$ restore option-letters corresponding-arguments [files-and-directories]
```

where *files-and-directories* is a list of files and directories for `restore` to retrieve from the backup tape. If no files are listed, the entire tape will be restored.

In the first, newer form, the first item is the list of options to be used for this backup with their arguments immediately following the option letters in the normal way (e.g., `-f /dev/tape`). In the second, older form, *option-letters* is a list of argument letters for the desired options, and *corresponding-arguments* are the values associated with each argument, in the same order. This syntax is still the only one available under AIX and Solaris.

Most options to `restore` do not have any arguments. However, as with `dump`, it is important that any arguments appear in the same order as the options requiring them.

`restore` places the files that it retrieves in the current working directory. When a directory is selected for restoration, `restore` restores the directory and all the files within it, unless you have specified the `-h` option (described later in this section).

`restore`'s most important options are the following:

`-r`
: *Read* and restore the entire tape. This is a very powerful command; it should be used only to restore an entire filesystem located on one or more tapes. The filesystem into which the tape is read should be newly created and completely empty. This option can also be used to restore a complete incremental dump on top of a newly restored filesystem. That is, after using the `-r` option to restore the most recent full dump, you use it again to restore successive incremental dumps until the filesystem has been completely restored.

`-x`
: *Extract* all files and directories listed and restore them in the current directory. Each filename to be extracted must be a complete pathname *relative* to the root directory of the filesystem being restored. For example, to restore the file */chem/pub/old/gold.dat* from a `dump` of the */chem* filesystem, you must specify the filename as *pub/old/gold.dat*. You should be in */chem* when you execute the `restore` command if you want the file to be restored to its original location.

`-t`
: *Type* the names of the listed files and directories if they appear on the backup tape. This option lets you find out whether a given file is on a particular tape more quickly than reading the entire tape. When used without a file list, it verifies that a `dump` tape is readable.

`-f` *`file`*
: The corresponding argument is the name of the file or device holding the dump. If this option is omitted, `restore` assumes that the dump tape is mounted on your default tape drive. Use a hyphen for *file* to specify standard input.

`-s` *`n`*
: The value *n* indicates which file on tape is to be used for the restore. For example, `-s 3` says to use the third tape file.

`-i`
: Enter *interactive* mode. This is almost always the most convenient method for restoring a small group of files. It is described in detail in the next section.

A typical usage of the `restore` command is:

```
# cd /home
# restore -x -f /dev/rmt1 chavez/mystuff others/myprogram
```

This restores the directory */home/chavez/mystuff* and the file called */home/others/myprogram* from a backup tape (assuming that */home* is the filesystem in the archive). The directories *chavez* and *others* are assumed to be in the current directory (and created if necessary), and the specified subdirectory and file are restored under them. These both originally resided within the */home* directory. Note, however, that the mount point name is not used in the `restore` command. The command must be executed from */home* to restore the files to their original locations.

On Solaris and HP-UX systems, the corresponding options would be:

```
xf /dev/rmt1 chavez/mystuff others/myprogram
```

`dump` and `restore` both save files independently of where the filesystem happens to be mounted at the time; that is, the pathnames used by these commands are relative to their position in their *own* filesystem, not in the overall system filesystem. This makes sense, because the filesystem could potentially be mounted anywhere in the overall directory tree, and files should still be able to be restored to their correct location relative to the current mount point for their filesystem.

If you need to restore some files that have been destroyed by accident, your most difficult problems will be determining which set of backup tapes contains these files and waiting for the system to read through one or more full backup tapes. If you do incremental backups, knowing when a file was last modified will help you to find the correct backup tape. Creating online table-of-contents files is also very useful (this topic is discussed later in this chapter).

The restore utility's interactive mode

The interactive mode is entered with `restore`'s `-i` option. Once there, the contents of a tape can be scanned and files chosen for extraction. This mode's use is illustrated in this sample session:

```
$ restore -i -f /dev/rmt1          Initiate restore's interactive mode.
restore > help
Available commands are:
   ls [arg] - list directory
   cd arg - change directory
   add [arg] - add `arg' to list of files to be extracted
   delete [arg] - delete `arg' from list of files to be extracted
   extract - extract requested files
...
If no `arg' is supplied, the current directory is used
restore > ls                       List directory on tape.
chavez/    harvey/     /ng
restore > cd chavez/vp             Change tape current directory.
restore > ls
v_a.c      v_a1.c      v_b3.c      v_d23.c      v_early
restore > add v_a1.c               Select (mark) files to be restored.
restore > add v_early
restore > ls
v_a.c     *v_a1.c      v_b3.c      v_d23.c     *v_early
restore > delete v_early           Remove a file from the extract list..
restore > extract                  Write selected files to current directory.
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 the first.
Specify next volume #: 1           Tape number if known.
set owner/mode for '.'? [yn] n     Don't change ./'s ownership or protection.
restore > quit                     End the restore interactive session.
```

The final prompt from `restore` asks whether to change the ownership and protection of the current directory to match that of the root directory on the tape. Answer yes only if you are restoring an entire filesystem.

Combining Several Backups onto a Single Tape

If you want to place several archives onto the same tape, all you need to do is rewind the tape (if necessary) before writing the first archive and then use a nonrewinding device for all subsequent backup operations.

To retrieve files from a multiarchive tape, you must position the tape at the proper location before issuing the restoration command. `restore` can do this automatically using its `-s` option, which takes the tape file number you want to use as its argument.

For all other backup types, position the tape with the `mt` command. For example, the following commands position the tape just after the second archive on the tape:

```
$ mt -f /dev/rmt0 rewind      If necessary
$ mt -f /dev/nrmt0 fsf 2
```

Again, you will need to use the nonrewinding form of the tape device; otherwise, the tape will be rewound to the beginning after positioning. Once at the desired point, you can write an additional backup archive to the tape or perform a restore operation using the next archive on the tape, as appropriate.

The HP-UX frecover utility

The HP-UX `frecover` utility restores files archived by `fbackup`, using a very similar syntax. For example, the first of the following commands restores the */chem/fullerenes* subdirectory tree:

```
# frecover -x -i /chem/fullerenes
# frecover -r -f /dev/rmt/1m
```

The second command restores all files on the tape in drive 1. `frecover` also accepts the `-i`, `-e`, and `-g` options. Other useful options include the following:

- `-X` and `-F` restore all retrieved files relative to the current directory (converting absolute pathnames to relative ones) or into the current directory (stripping off all paths), respectively.
- `-o` says to overwrite files on disk that are newer than the file in the backup set.
- `-N` says to read the backup media without restoring any files. It is useful for verifying the integrity of a backup and for creating table-of-contents files.

Moving Data Between Systems

In general, `tar`, `cpio`, and `dump` write archives that are readable on many different computer systems. However, sometimes you will run into problems reading a tape on a system other than the one on which it was written. There are four major causes for such problems:

Block size differences

The simplest cause of tape reading problems is a difference in the block size with which the archive was written and the block size expected by the drive on which you're trying to read it. Some tape drives assume specific fixed block sizes. You can specify the block size to backup and restore utilities (`-b` is often the relevant option), and on many systems you can set the characteristics of the drive itself. The most commonly used block sizes are 512 and 1024.

Archive format incompatibilities
: The backup utilities provided by early versions of Unix differed from those in use today, so very old computer systems may not be able to read tapes written on a current machine. The modern versions of most utilities include backward-compatibility options that allow you to write tapes in the old format if you need to read them on an ancient system.

Byte order differences
: Whether a computer system is *big endian* or *little endian* determines how it interprets the individual bytes within larger data units, such as words. Big-endian systems consider the byte with the lowest address as the most significant; little-endian systems consider it to be the least significant. Tape archives, like all other data on a computer system, reflect this fundamental attribute of the hardware. When you want to read a tape produced by a computer of one type on a different computer of the other type, you'll need to swap the bytes before utilities like `tar` can make sense of the archive.

For example, you could use this AIX command to list the contents of a tape written on an IRIX system:

```
$ dd if=/dev/rmt1 conv=swab | tar tvf -
```

The `dd` command reads the tape file and swaps the bytes, passing the converted archive to the `tar` command, which lists the archive it finds on standard input. You could construct the equivalent reversed pipe to produce a byte-swapped archive on tape.

Compressed archives
: If you write a tape on a drive that performs automatic data compression, you won't be able to read it on a drive that lacks this feature. In order to write tapes that will be readable on drives without compression, you'll need to specify the noncompressing special file to the backup utility (refer to the discussion of special files earlier in this chapter, as well as the relevant manual pages for your systems, for details).

Making Table of Contents Files

It is often convenient to have online listings of the contents of system backup tapes. For one thing, they make it much easier to figure out which tape has the file you need to restore, especially when multiple levels of incremental backups are in use. It is quite easy to create such files at the time the backup is performed.

If you're using tar or cpio for backup, you can take advantage of the -v option to create a listing of the tape's contents as it is written, as in these examples:

```
$ today='date +%d%b%Y'
$ tar -cv /home > /backup/home_full_$today.TOC
                        or
$ tar -cv /home | tee /backup/home_full_$today.TOC
```

Both tar commands archive the contents of */home*, generating a long, directory-like listing as it does so and saving it to a file with a name like */backup/home_full_21mar1995.TOC*. The second command also displays the same output on the screen.

cpio sends the file list to standard error, so it must be captured slightly differently:

```
$ toc='date +/backup/home_full_%d%b%y.TOC'
$ find /home -print |  cpio -ov > /dev/rmt0 2> $toc
```

If you want to use the C shell, the commands are a little different:

```
% set toc='date +/backup/home_full_%d%b%y.TOC'
% (find /home -print | cpio -ov > /dev/rmt0) >& $toc
```

The file lists produced by cpio commands like these contain only the pathnames of the files in the archive. If you want a more detailed listing, you can generate it with a second cpio command or a more complex pipe leading up to the cpio backup command:

```
$ cpio -itv < /dev/rmt0 > $toc
$ find /home | cpio -o | tee /dev/rmt0 | cpio -t -i -v > $toc
```

The first command lists the files in the archive on tape. The second command avoids having to reread the backup tape by using the find command to generate a list of files, which cpio makes into an archive. This archive is then sent both the the tape drive and to another cpio command. The latter lists the archive contents and writes it to the specified table-of-contents file.

Making a table of contents file for a dump tape requires a subsequent restore command. For example, here is a script that performs a backup with dump and then creates a table-of-contents file with restore:

```
#!/bin/csh
# bkup+toc - perform dump and verify tape/make TOC file
# $1 = filesystem
# $2 = dump level (default=0)
#
if ($#argv < 1) then
  echo "do_backup: filesystem [dump-level]"
  exit 1
endif

set lev=0
if ("$2" != "") set lev=$2
dump -${lev} -u -f /dev/rmt1 $1
if ($status) then
```

```
      echo "do_backup: dump failed"
      exit 1
    endif
    restore -t -v -f /dev/rmt1 > /backup/`date +$1:t_%m-%d-%Y.$lev`
```

This script runs the dump command on the filesystem given as its first argument, using the backup level specified as its second argument (or level 0 by default). If the dump command exits normally, the restore command is used to verify the backup and write its contents to a file. The file's name contains the disk name and the month, day, and year when the backup was done, and its extension is the backup level: e.g., *chem_06-24-2001.2* would be the filename for a level 2 backup of */chem* made on June 24, 2001.

On an HP-UX system, you can use this frecover command to create a table-of-content file:

```
    # frecover -r -Nv -f /dev/rmt/0m > $toc
```

Network Backup Systems

So far, we've considered only backups and restores of disks on a local computer system. However, many organizations need to take a more unified and comprehensive approach to their total backup needs. We will consider various available solutions for this problem in this section.

Remote Backups and Restores

The simplest way to move beyond the single-system backup view is to consider remote backup and restores. It is very common to want to perform a backup over the network. The reasons are varied: your system may not have a tape drive at all since not all systems come with one by default any more, there may be a better (faster, higher capacity) tape drive on another system, and so on.

Most versions of dump and restore can perform network-based operations (Tru64 requires you to use the separate rdump and rrestore commands). This is accomplished by specifying a device name of the form *host:local_device* as an argument to the -f option. The hostname may also optionally be preceded by a username and at-sign; for example, -f chavez@hamlet:/dev/rmt1 performs the operation on device */dev/rmt1* on host *hamlet* as user *chavez*.

This capability uses the same network services as the rsh and rcp commands. Remote backup facilities depend on the daemon */usr/sbin/rmt* (which is often linked to */etc/rmt*).[*] To be allowed access on the remote system, there needs to be a *.rhosts* in its root directory, containing at least the name of the (local) host from which the data will come. This file must be owned by *root*, and its mode must not allow any

[*] On a few older systems, you'll need to create the link yourself.

access by group or other users (for example, 400). This mechanism has the mechanism's usual negative security implications (see "Network Security" in Chapter 7).

Some versions of the tar command can also use the rmt remote tape facility.

The HP-UX fbackup and frestore utilities accept remote tape drives as arguments to the normal -f option. For example:

```
# fbackup -0u -f backuphost:/dev/rmt/1m -i /chem
```

The Amanda Facility

Amanda is the Advanced Maryland Automated Network Disk Archiver. It was developed at the University of Maryland (James da Silva was the initial author). The project's home page is *http://www.amanda.org*, where it can be obtained free of charge. This section provides an overview of Amanda. Consult Chapter 4 of *Unix Backup and Recovery* for a very detailed discussion of all of Amanda's features (this chapter is also available on the Amanda home page).

About Amanda

Amanda allows backups from a network of clients to be sent to a single designated backup server. The package operates by functioning as a wrapper around native backup software like GNU tar and dump. It can also back up files from Windows clients via the Samba facility (smbtar). It has a number of nice features:

- It uses its own network protocols and thus does not suffer from the security problems inherent in the rmt approach.
- It supports many common tape and other backup devices (including stackers and jukeboxes).
- It can perform full and incremental backups and decide the backup level automatically based on specified configuration parameters.
- It can take advantage of hardware compression features, or it can compress archives prior to writing them to tape (or other media) when the former is not available. Software compression may be performed either by the main server or by the client system.
- It provides excellent protection against accidental media overwriting.
- It can use holding disks as intermediate storage for backup archives to maximize tape write performance and to ensure that data is backed up in spite of tape errors (allowing the backup set to be written to backup media at a later time).
- It can use Kerberos-based authentication in addition to providing its own authentication scheme. Kerberos encryption can also be used to protect the data as it is transmitted across the network.

At present, Amanda does have a couple of annoying limitations:

- It cannot split a backup archive across multiple tapes. When it encounters an end-of-tape mark while saving a backup archive, it begins writing the archive from the beginning on the next tape.
- It cannot produce individual backup archives larger than a single tape. This is a consequence of the first limitation.
- Only a single backup server is supported.

How Amanda works

Amanda uses a combination of full and incremental backups to save all of the data for which it is responsible, using the smallest possible daily backup set that can do so. Its scheme first computes the total amount of data to be backed up. It uses this total, along with a couple of parameters defined by the system administrator, to figure out what to do in the current run. These are the key parameters:

The number of runs in a backup cycle
: At a rate of one Amanda run per day, this corresponds to the desired number of days between full backups.

The percentage of data that changes between Amanda runs
: In the single run per day case, this is the percentage of the data that changes each day.

Amanda's overall strategy is twofold: to complete a full backup of the data within each cycle and to be sure that all changed data has been backed up between full dumps. The traditional method of doing this is to perform the full backup followed by incrementals on the days between them. Amanda operates differently.

Each run (night), Amanda performs a full backup of part of the data, specifically, the fraction that is required to back up the entire data set in the course of a complete backup cycle. For example, if the cycle is 7 days long (with one run per day), 1/7 of the data must be backed up each day to complete a full backup in 7 days. In addition to this "partial" full backup, Amanda also performs incremental backups for all data that has changed since its own last full backup.

Figure 11-1 illustrates an Amanda backup cycle lasting 4 days, in which 15% of the data changes from day to day. The box at the top of the figure stands for the complete set of data for which Amanda is responsible; we have divided it into four segments to represent the part of the data that gets a full backup at the same time.

The contents of the nightly backups are shown at the bottom of the figure. The first three days represent a start-up period. On the first night, the first quarter of the data is fully backed up. On the second night, the second quarter is fully backed up, and the 15% of the data from the previous night that changed during day 2 is also saved. On day 3, the third quarter of the total data is fully backed up, as well as the changed 15% of day 2's backup. In addition, 15% of the portion backed up on the first night

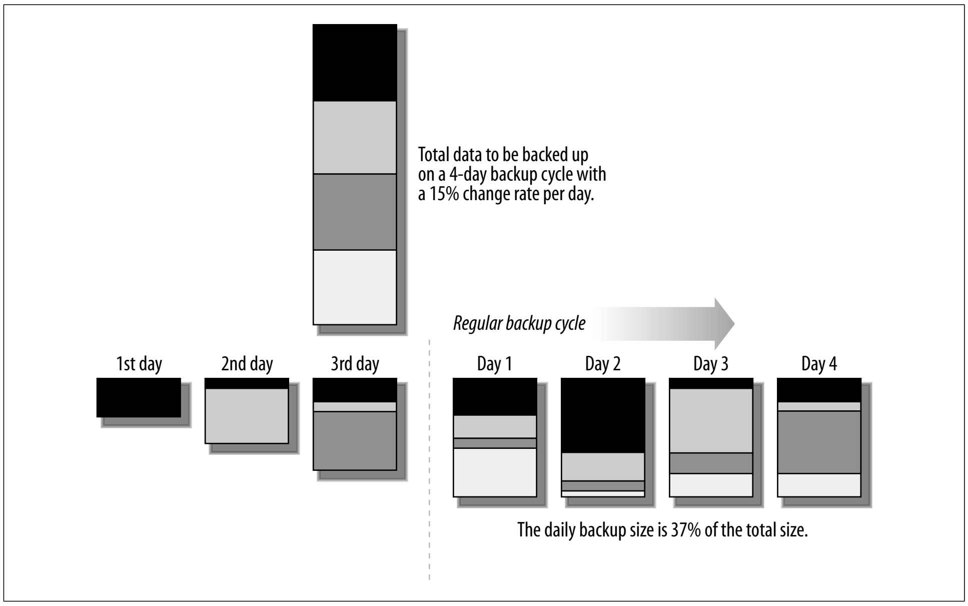

Total data to be backed up on a 4-day backup cycle with a 15% change rate per day.
Regular backup cycle
1st day
2nd day
3rd day
Day 1
Day 2
Day 3
Day 4
The daily backup size is 37% of the total size.

Figure 11-1. The Amanda backup scheme

is written for each of the intervening nights since its full backup: in other words, 30% of that quarter of the total data.

By day 4, the normal schedule is in force. Each night, one quarter of the total data is backed up in full, and incrementals are performed for each of the other quarters as appropriate to the time that has passed since their last full backup.

This example uses only first-level incremental backups. In actual practice, Amanda uses multiple levels of incremental backups to minimize backup storage requirements.

To restore files from an Amanda backup, you may need one complete cycle of media.

Let's now consider a numeric example. Suppose we have 100 GB of data that we need to back up. Table 11-3 illustrates four Amanda backup schedules based on differing cycle lengths and per-day change percentages.

Table 11-3. Sample Amanda backup sizes (total data=100 GB)

	3-day cycle 10% change	5-day cycle 10% change	7-day cycle 10% change	7-day cycle 15% change
Full portion	33.3	20.0	14.3	14.3
1st previous day	3.4	2.0	1.4	2.2
2nd previous day	6.8	4.0	2.8	4.4
3rd previous day		6.0	4.2	6.6

Table 11-3. Sample Amanda backup sizes (total data=100 GB) (continued)

	3-day cycle 10% change	5-day cycle 10% change	7-day cycle 10% change	7-day cycle 15% change
4th previous day		8.0	5.6	8.8
5th previous day			7.0	11.0
6th previous day			8.4	13.2
Daily size (GB)	**43.5**	**40.0**	**43.7**	**60.5**

The table columns illustrate the data that would comprise each daily backup, breaking it down by the full backup portion and the incremental data from each previous full backup within the cycle.

Note that Amanda computes what should be backed up every time it is run, so it is not as static as the preceding examples suggest, but the examples nevertheless provide a general picture of how the facility operates.

In the next section, we consider how the backup size depends on the backup cycle more formally, including some expressions that can be used to decide on an appropriate backup cycle for specific conditions.

Estimating the Daily Change Rate

You can use the `find` command to help estimate the daily change rate:

```
$ find dir -newer /var/adm/yesterday -ls | \
  awk '{sum+=$7}; END {print "diff =",sum}'
```

Repeat the command as needed to cover all the data to be backed up. Use `touch` to update the time for the file */var/adm/yesterday* after all the `find` commands are run.

Then, divide this value by the total used space (e.g., taken from `df` output). Repeat the process for several days or weeks to determine an average rate.

Doing the math

Next, we consider some expressions that can be used to compute starting parameters for Amanda (which can be fine-tuned over time, based on actual use). If this sort of mathematical analysis is of no interest to you, just skip this section.

We will use the following variables:

T = total amount of data
p = percentage change between runs (in decimal form: e.g. 12%=0.12)
n = number of runs in a complete cycle (often days)
S = amount of data that must be backed up every run (day)
F = fraction of the total data that must be backed up every run (day): S/T

To compute per-run amount of data that must be backed up, use this expression for S:

$$S = \frac{T}{n} + \frac{Tp(n-1)}{2}$$

For example, 70 GB of data that changes by 10% per day using a 1 week backup cycle requires that 31 GB be backed up every night ($70/7 + 70 \times 0.1 \times 6/2 = 10 + 42/2 = 10 + 21 = 31$). If 31 GB is larger than the maximum capacity that you have in the available time, you'll need to adjust the other parameters (see below).

Alternatively, if you have a fixed amount of backup capacity per run, you can figure out the required cycle length. Refer to the discussion of capacity planning earlier in this chapter for information on determining how much capacity you have.

To compute *n* for a given nightly capacity, use this expression:

$$n = \frac{x \pm \sqrt{x^2 - 2p}}{p}$$

where

$$x = \left(\frac{p}{2} + \frac{S}{T}\right)$$

We have introduced the variable *x* to make the expression for *n* simpler. Suppose that you have a nightly backup capacity of 40 GB for the same scenario (70 GB total data, changing at 10% per day). Then $x = 0.1/2 + 40/70 = 0.05 + 0.57 = 0.62$. We can now compute *n*: $(0.62 \pm \sqrt{0.38 - 0.2}) / 0.1 = (0.62 \pm \sqrt{0.18}) / 0.1 = (0.62 \pm 0.42) / 0.1 = 6.2 \pm 4.2$.

This calculation yields solutions of 2 and 11 (rounding to integers). We can either do full backups of about half the data every night or use a much longer 11-day cycle and still be able to get the backups all done. Note that these values take maximum advantage of the available capacity.

Now suppose that you have a nightly backup capacity of only 20 GB for the same scenario (70 GB total data, changing at 10% per day). Then $x = 0.1/2 + 40/70 = 0.05 + 0.29 = 0.34$. We can now compute *n*: $(0.34 \pm \sqrt{0.12 - 0.2}) / 0.1$. The square-root term is now imaginary (since 0.12–0.20 is negative), indicating that this proposed configuration will not work in practice.* The available capacity is simply too small.

In general, you can compute the minimum per-run capacity for a given per-run percentage change (*p*) with this expression (which introduces *F* as the fraction of the total data that must be backed up):

$$F_{minimum\ for\ fixed\ p} = 2\sqrt{\frac{p}{2}} - \frac{p}{2} \text{ (where } F = \frac{S}{T}\text{);} \quad \therefore S_{minimum} = FT$$

* Mathematically, there are no real solutions to the underlying quadratic equation.

F indicates the fraction of that data that must be backed up each run in order for the system to succeed. So, in our case of a 10% change rate, $F = 2 \times \sqrt{0.1/2} - (0.1/2) = 2 \times \sqrt{0.05} - 0.05 = 2 \times 0.22 - 0.05 = 0.44 - 0.05 = 0.39 \approx 40\%$. Note that this expression is independent of *T* (the total backup data); whenever the data changes by about 10% per run, you must be able to back up at least 40% of the total data every run for success. In our case, this corresponds to a minimum nightly capacity of $0.4 \times 70 = 29$ GB.

Alternatively, you can compute the run cycle *n* that is required to minimize *F* (and thus *S*) for a given value of *p* with this expression:*

$$n_{minimum\ S} = \sqrt{\frac{2}{p}}$$

In our case, the cycle period which minimizes the amount of data to be backed up is $\sqrt{2/0.01} = \sqrt{20} = 4.47 \approx 5$ days. Again, this value is independent of the amount of data. In our case, when the data is changing by 10% per day, a cycle time of 5 days will minimize the amount of data that must be backed up every night. This is the most efficient cycle length with the minimum nightly backup capacity.

Thus, both the minimum time cycle and per-run fraction of data to back up are determined only by the rate at which the data is changing, and the actual per-run backup size for a given amount of total backup data can be easily computed from them. Thus, having an accurate estimate for *p* is vital to rational planning.

This discussion ignores compression in analyzing backup procedures. If your tape drive can compress data, or if you decide to compress it with software before writing it to tape, you will need to take the expected compression factor into account in your computations.

Configuring Amanda

Building and installing Amanda is generally straightforward, and the process is well-documented, so we will not consider it here.

The Amanda system includes the following components:

- Client programs, of which `amandad` is the most important. This daemon communicates with the Amanda server during backup runs, calling other client programs as appropriate: `selfcheck` (verify local Amanda configuration), `sendsize` (estimate backup size), `sendbackup` (perform backup operations), and `amcheck` (verify Amanda setup). These programs are part of the Amanda client system; on the Amanda server, these programs are found with the package's other helper programs, in */usr/local/lib/amanda* or */usr/lib/amanda*.

* Mathematically, the value of *n* where $\partial F / \partial n = 0$. In this specific example, the mathematical region around the minimum is quite flat.

- Server programs to perform the various phases of the actual backup operations. The amdump program is the one that initiates an Amanda run, and it is usually run periodically from cron. It controls a number of other programs, including planner (determine what to backup), driver (interface to device), dumper (communicate with client amandad processes), taper (write data to media), and amreport (prepare report for an Amanda run).
- Administrative utilities to perform related tasks. They include amcheck (verify Amanda configuration is valid and the facility is ready to run), amlabel (prepare media for use with Amanda), amcleanup (clean up after an aborted run or system crash), amflush (force data from the holding area to backup media), and amadmin (perform various administrative functions).
- Configuration files that specify Amanda operations, such as what to back up and how often to do so, as well as the locations and characteristics of the tape device. These files are *amanda.conf* and *disklist*, and they reside in a subdirectory of the main Amanda directory (canonically, this location is */usr/local/etc/amanda*, but it can be */etc/amanda* when the package is preinstalled). A typical name is *Daily*. Each subdirectory corresponds to an Amanda "configuration," a distinct set of settings and options referred to by the directory name.
- The amrestore utility, which can be used to restore data from Amanda backups. In addition, the amrecover utility supports interactive file restoration. It relies on a couple of daemons to do its job: amindexd and amidxtaped.

Setting up an Amanda client. Once you have installed the Amanda software on a client system, there are a few additional steps to take. First, you must add entries to the */etc/inetd.conf* and */etc/services* files to enable support for the Amanda network services:

/etc/services:
```
amanda    10080/udp
```

/etc/inetd.conf:
```
amanda  dgram   udp   wait   amanda   /path/amandad   amandad
```

The Amanda daemon runs as user *amanda* in this example; you should use whatever username you specified when you installed the Amanda software.

In addition, you'll need to ensure that all the data that you want to be backed up is readable by the Amanda user and group. Similarly, the file */etc/dumpdates* must exist and be writeable by the Amanda group.

Finally, you must set up the authorization scheme that amandad will use. This is usually selected at compile time. You may use normal *.rhosts*-based authentication, Kerberos authentication (see below) or a separate *.amandahosts* (the default mechanism). The *.amandahosts* file is similar to a *.rhosts* file, but it applies only to the Amanda facility and so does not carry the same level of risk. Consult the Amanda documentation for full information about authentication options.

Selecting an Amanda server. Selecting an appropriate system as the Amanda server is crucial to good performance. You should keep the following items in mind:

- The system should have the best tape drives (or other backup devices) possible.
- The system should have sufficient network bandwidth for the estimated data flow.
- The system should have sufficient disk space for the holding area. A good size is at least twice the size of the largest per-run dump size.
- If the server will be performing software compression on the data, a fast CPU is necessary.
- Large amounts of memory will have little effect on backup performance, so there is no reason to overconfigure the system with memory.

Setting up the Amanda server. There are several steps necessary to configure the Amanda server once the software is installed. First of all, you must add entries to the same network configuration files as those for Amanda clients:

```
/etc/services:
amanda        10080/udp
amandaidx     10082/tcp
amidxtape     10083/tcp

/etc/inetd.conf:
amandaidx  stream  tcp  nowait  amanda /path/amindexd   amindexd
amidxtape  stream  tcp  nowait  amanda /path/amidxtaped amidxtaped
```

Next, you must configure Amanda by creating the required configuration files. Create a new subdirectory under *etc/amanda* in the top-level Amanda directory (i.e., */usr/local* or */*), if necessary. We will use *Daily* as our example. Then, create and modify *amanda.conf* and *disklist* configuration files in this subdirectory (the Amanda package contains example files that can be used as a starting point).

We will begin with *amanda.conf* and consider its contents in groups of related entries. We will examine an annotated sample *amanda.conf* file.

The initial entries in the file typically specify information about the local site and locations of important files:

```
org "ahania.com"              Organization name for reports.
mailto "amanda-rep"           Mail reports to this user.
dumpuser "amanda"             Amanda user account.
printer "tlabels"             Printer for tape labels.
logdir "/var/log/amanda"      Put log files here.
indexdir "/var/adm/amindex"   Store backup set index data here.
```

The next few entries specify the basic parameters for the backup procedure:

```
# fundamental parameters
dumpcycle 7 days     Length of the backup cycle (default=10 days).
runspercycle 5       Amanda runs per cycle (if < 1/day).
```

```
# network-related resource settings
netusage 400 kps        Maximum network bandwidth (default=300).
inparallel 20           Max. simultaneous backups (default=10).
ctimeout 120            Client timeout period (default=30 seconds).

# incremental level bump parameters
bumpsize 20 mb          Min. savings for level 2 incrs. (default=10).
bumpdays 1              Required # days at each level (default=2).
bumpmult 2              Multiply bumpsize by this for each higher incremental level (default=1.5).
```

The incremental bump level parameters specify when Amanda should increase the incremental backup level in order to make the backup set size smaller. Using these settings, Amanda will switch from level 1 incrementals to level 2 incrementals whenever it will save at least 20 MB of space. The multiplication factor has the effect of requiring additional savings to move to each higher incremental level. The threshold for each level is this factor times the saving required for the previous level, i.e., 40 for levels 2 to 3, 80 for levels 3 to 4, and so on. This strategy is designed to ensure that the added complexity of multiple levels of incremental backups also bring significant savings in the size of the backup set.

These next entries specify information about the tape drive and media to use:

```
# number of tapes in use          Set to at least # tapes required for one full cycle
tapecycle 25                        plus a few spares (default=15).
labelstr "Daily[0-9][0-9]*"       Format of the table labels (regular expression).

tapedev "/dev/rmt/0"
tapetype "DLT"

#changerdev "/dev/whatever"
#tpchanger "script-path"          Script to change to next tape (supplied).
#runtapes 4                         Maximum number of tapes per run.
```

The first two entries specify the number of tapes in use and the pattern used by their electronic labels. Note that tapes must be prepared with `amlabel` prior to use (discussed below).

The next two entries specify the location of the tape drive and its type. The final three entries are used with tape changers and are commented out in this example. Only one of *tapedev* and *tpchanger* must be used.

Tape types are defined elsewhere in the configuration file with stanzas like this:

```
define tapetype DLT {
    comment "DLT with 10 GB tapes"
    length 12500 mb      Tape capacity (takes compression into account).
    speed 1536 kps       Drive speed.
    lbl-templ "file"     PostScript template file for printed labels.
}
```

The example configuration file includes many defined tape types. The *length* and *speed* parameters are used only for estimation purposes (e.g., how many tapes will be required). When performing the actual data transfer to tape, Amanda will keep writing until it encounters an end-of-tape mark.

The following entry and *holdingdisk* stanza defines a disk holding area:

```
# When media is unavailable, save this % of holding space
# for degraded-mode incremental backups.
reserve 50                Default is 100%.

holdingdisk amhold0 {     Name is amhold0.
   comment "Primary holding disk"
   directory "/scratch/amanda"
# amount of space to use (+) or save (-); 0=use all (default)
   use -2 Gb              Always leave this much space.
}
```

More than one holding disk may be defined.

The final task to be done in the configuration file is to define various dump types: generalized backup actions having specific characteristics (but independent of the data to be backed up). Here is an example for the *normal* backup type (you can choose any names you like):

```
define dumptype normal {
   comment "Ordinary backup"
   holdingdisk yes      Use a holding disk.
   index yes            Maintain index info on contents.
   program "DUMP"       Backup command.
   priority medium      Specify backup relative priority.
#  use 24-hour clock without punctuation
   starttime 2000       Don't begin backup before this time (8 P.M. here).
}
```

This dump type uses a holding disk, creates an index for the backup set contents for interactive restoration and uses the dump program to perform the actual backup. It runs at medium priority compared to other backups (the possibilities are *low* (0), *medium* (1), *high* (2) and an arbitrary integer, with higher numbers meaning the backup will be performed sooner). Backups using this method will not begin before 8 pm regardless of when the amdump command is issued.

Amanda provides several pre-defined dump types in the example *amanda.conf* file which can be used or customized as desired.

Here are some other parameters that are useful in dump type definitions:

```
program "GNUTAR"          Use the GNU tar program for backups.
                          This is also the value to use for Samba backups.
exclude ".exclude"        GNU tar exclusion file (located in top-level
                          of the filesystem to be backed up).
compress server "fast"    Use software compression on server using the
                          fastest compression method. Other keywords are
                          "client" and "best".
auth "krb4"               Use Kerberos 4 user authentication.
kencrypt yes              Encrypt transmitted data.
ignore yes                Do not run this backup type.
```

Amanda's *disklist* configuration file specifies the actual filesystems to be backed up. Here are some sample entries:

```
# host     partition      dumptype    spindle
hamlet      sd1a           normal        -1
hamlet      sd2a           normal        -1
dalton      /chem          srv_comp      -1
leda        //leda/e       samba         -1   # Win2K system
astarte     /data1         normal         1
astarte     /data2         normal         1
astarte     /home          normal         2   # dump all alone
```

The columns in this file hold the hostname, disk partition (specified by file in */dev*, full special file name, or mount point), the dump type, and a spindle parameter. The latter serves to control which backups can be done at the same time on a host. A value of -1 says to ignore this parameter. Other values define backup groups within a host; Amanda will only run backups from the same group in parallel. For example, on host *astarte*, the */home* filesystem must be backed up separately from the other two (the latter may be backed up simultaneously if Amanda so wishes).

There are a few final steps that are needed to complete the Amanda server setup:

- Prepare media with the `amlabel` command. For example, the following command will prepare a tape labeled "DAILY05" for use with the Amanda configuration named *Daily*:

  ```
  $ amlabel Daily DAILY05
  ```

  Similarly, the following command will prepare the tape in slot 5 of the associated tape device as "CHEM101" for use with the *Chem* configuration:

  ```
  $ amlabel Chem CHEM101 slot 5
  ```

- Use the `amcheck` command to check and verify the Amanda configuration.
- Create a `cron` job for the Amanda user to run the `amdump` command on a regular basis (e.g., nightly). This command takes the desired configuration as its argument.

Amanda expects the proper tape to be in the tape drive when the backup process begins. You can determine the next tape needed for the *Daily* configuration by running the following command:

```
# amadmin Daily tape
```

The Amanda system will need some ongoing administration, including tuning and cleanup. The latter is accomplished via the `amflush` and `amcleanup` commands. `amflush` is used to force the data in the holding disk to backup media, and it is typically required after a media failure occurs during an Amanda run. In such cases, the backup data is still written to the holding disk. The `amcleanup` command needs to be run after an Amanda run aborts or after a system crash.

Finally, you can temporarily disable an Amanda configuration by creating a file named *hold* in the corresponding subdirectory. While this file exists, the Amanda system will pause. This can be used to keep the configuration information intact in the event of a hardware failure on the backup device or a device being temporarily needed for another task.

Amanda reports and logs

The Amanda system produces a report for each backup run and sends it by electronic mail to the user specified in the *amanda.conf* configuration file. The reports are quite detailed and contain the following sections:

- The dump date and time and estimated media requirements:

  ```
  These dumps were to tape DAILY05.
  Tonight's dumps should go onto one tape: DAILY05.
  ```

- A summary of errors and other aberrations encountered during the run:

  ```
  FAILURE AND STRANGE DUMP SUMMARY:
  dalton.ahania.com /chem lev 0 FAILED [request ... timed out.]
  ```

  Host *dalton* was down so the backup failed.

- Statistics about the run, including data sizes and write rates (output has been shortened):

  ```
  STATISTICS:
                            Total      Full      Daily
                          --------  --------   --------
  Dump Time (hrs:min)       2:48      2:21       0:27
  Output Size (meg)       9344.3    7221.1     2123.2
  Original Size (meg)     9344.3    7221.1     2123.2
  Avg Compressed Size (%)   --        --         --
  Tape Used (%)             93.4      72.2       21.2
  Filesystems Dumped          10         2          8
  Avg Dump Rate (k/s)     1032.1    1322.7      398.1
  Avg Tp Write Rate (k/s) 1234.6    1556.2     1123.8
  ```

- Additional information about some of the errors/aberrations, when available.
- Informative messages from the various subprograms called by `amdump`:

  ```
  NOTES:
     planner: Adding new disk hamlet.ahania.com:/sda2
     taper: tape DAILY05 9568563 kb fm 1 [OK]
  ```

- A summary table listing the data that was backed up and related information:

  ```
  DUMP SUMMARY:
                          DUMPER STATS             TAPER STATS
  HOST   DISK  L ORIG-KB OUT-KB COMP% MMM:SS  KB/s MMM:SS  KB/s
  ------------------------------------------------------------
  hamlet sd1a  1   28255  28255   --    2:36 180.3  0:21 1321.1
  hamlet sd2a  0  466523 466523   --   36:51 211.1  5:33 1400.8
  dalton /chem 1  FAILED-------------------------------------
  ada    /home 1   39781  39781   --    5:16 125.7  0:29 1356.7
  ...
  ```

You should examine the reports regularly, especially the sections related to errors and performance.

Amanda also produces log files for each run, *amdump.n*, and *log.date.n*, located in the designated log file directory. These are more verbose versions of the email report, and they can be helpful in tracking some sorts of problems.

Restoring files from an Amanda backup

Amanda provides the interactive amrecover utility for restoring files from Amanda backups. It requires that backup sets be indexed (using the *index yes* setting) and that the two indexing daemons mentioned previously be enabled. The utility must be run as *root* from the appropriate client system.

Here is a sample session:

```
# amrecover Daily
AMRECOVER Version 2.4.2. Contacting server on depot.ahania.com ...
...
Setting restore date to today (2001-08-12)
200 Working date set to 2001-08-14.
200 Config set to Daily.
200 Dump host set to astarte.ahania.com.
$CWD '/home/chavez/data' is on disk '/home' mounted at '/home'.
200 Disk set to /home.
amrecover> cd chavez/data
/home/chavez/data
amrecover> add jetfuel.jpg
Added /chavez/data/jetfuel.jpg
amrecover> extract
Extracting files using tape drive /dev/rmt0 on host depot...
The following tapes are needed: DAILY02
Restoring files into directory /home
Continue? [Y/n]: y
Load tape DAILY02 now
Continue? [Y/n]: y
warning: ./chavez: File exists
Warning: ./chavez/data: File exists
Set owner/mode for '.'? [yn]: n
amrecover> quit
```

In this case, the amrecover command is very similar to the standard restore command in its interactive mode.

The amrestore command can also be used to restore data from an Amanda backup. It is designed to restore entire images from Amanda tapes. See its manual page or the discussion in *Unix Backup and Restore* for details on its use.

Commercial Backup Packages

There are several excellent commercial backup facilities available. An up-to-date list of current packages can be obtained from *http://www.storagemountain.com.* We won't consider any particular package here but, rather, briefly summarize the important features of a general-purpose backup package, which can potentially serve as criteria for comparing and evaluating any products your site is considering.

You should expect the following features from a high-end commercial backup software package suitable for medium-sized and larger networks:

- The ability to define backups sets as arbitrary lists of files that can be saved and reloaded into the utility as needed.
- A capability for defining and saving the characteristics and data comprising standard backup operations.
- A facility for exclusion lists, allowing you to create, save, and load lists of files and directories to exclude from a backup operation (including wildcard specifications).
- An automated backup scheduling facility accessed and controlled from within the backup utility itself.
- The ability to specify default settings for backup and restore operations.
- The ability to back up all important file types (e.g., device files, sparse files) and attributes (e.g., access control lists).
- The ability to back up open files or to skip them entirely without pausing (at your option).
- The ability to define and initiate remote backup and restore operations.
- Support for multiple backup servers.
- Support for high-end backup devices, such as stackers, jukeboxes, libraries and silos.
- Support for tape RAID devices, in which multiple physical tapes are combined into a single high-performance logical unit via parallel write operations.
- Support for non-tape backup devices, such as removable disks.
- The capability to perform multiple operations to distinct tape devices simultaneously.
- Support for multiplexed backup operations in which multiple data streams are backed up to a single tape device at the same time.
- Support for clients running all of the operating systems in use at your site.
- Compatibility with the standard backup utilities, which may be important to some sites (so that saved files can be restored to any system).
- Facilities for automatic archiving of inactive files to alternate online storage devices (for example, jukeboxes of optical disks) to conserve disk space and reduce backup requirements.
- Inclusion of some kind of database manager so that you (and the backup software) can perform queries to find the media needed to restore files.

See Chapter 5 of *Unix Backup and Recovery* for an extended discussion of commercial backup package features.

Backing Up and Restoring the System Filesystems

This final section covers backing up and restoring the filesystem containing the operating system itself, including the case of a system disk failure. Recovering from such a disaster has come to be known as "bare metal recovery" in recent years. *Unix Backup and Restore* includes detailed chapters describing these techniques and procedures for several Unix varieties.

Filesystems containing operating system files such as / and */usr* pose few problems when all you need to restore is the occasional accidentally deleted or otherwise lost file. When the file in question is an unmodified system file, you can usually restore it from the operating system installation media, provided you have it and that it is readable under normal system conditions. If either of these conditions is not true, you should do a full backup of all system filesystems from time to time.

Files that you modify in the system partitions should be backed up regularly. In Chapter 14, we looked at a script that saves all modified configuration and other files to a user filesystem, allowing them to be backed up regularly and automatically via the system backup procedures. Alternatively, the script could save them directly to the backup media (even to a diskette if the archive is small enough).

When system filesystems need to be completely restored (usually due to hardware problems), some special considerations come into play. There are often two distinct approaches that can be taken:

- Reinstalling from the original operating system installation tapes or CDs and then restoring files you have modified. This approach may also involve reconfiguring some subsystems.
- Booting from alternate media and then restoring the filesystems from full backups that you have made.

Which alternative is preferable depends a lot on the characteristics of your particular system: how many files have been customized and how widely they are spread across the various system filesystems, how much device and other reconfiguration needs to be redone, and similar considerations. If you have to restore multiple partitions, it is usually faster to reinstall the operating system from scratch unless unsaved data in another partition on the same disk will be lost using the standard installation procedures.

If you decide to take the second route, booting from alternate media and then restoring from a backup, you will need to make reliable full backups of the system whenever it changes significantly. Because you are depending on them for a system restoration in an emergency, these backups should be verified or even made in duplicate.

In either case, you will sometimes also need to consult records of the disk partitions and associated filesystem layouts, as well as the logical volume configuration, when a logical volume manager is in use. This is vital when the system disk has been damaged and must be replaced to restore the system to its previous configuration. Be sure to keep records of this data (see below).

Here is a general procedure for restoring a key filesystem from a backup (some of the individual steps are discussed in detail Chapter 10):

- Boot off alternate media, either an installation tape or CD, or a special bootable diskette or tape (discussed in a bit). At this point, you will be running off an in-memory filesystem (RAM disk) or one based on the boot medium.
- Create device files for the disks, disk partitions, and/or tape drive that you will need to access, if necessary. They may already have been provided for you if you used a system utility to create the bootable tape or diskette.
- Prepare the hard disk as necessary. This may include formatting (rarely) or partitioning it. Be sure to do anything required to make the disk bootable.
- Create a new filesystem on the appropriate partition, if necessary.
- Mount the system filesystem (*/mnt* is the conventional location).
- Change the current directory to the mount point. Restore the files from the backup tape. Afterwards, change back to the root directory and dismount the restored filesystem.
- Repeat the process for any additional filesystem and then reboot the system.

There is one additional point to consider when using this approach—or planning to rely on it. The filesystem provided by emergency boot tapes or diskettes is very limited, and only a small subset of the normal system commands are available. You will need to verify that the restoration utility you need is available after booting from alternate media. For example, if the boot diskette provides only `cpio`, the backup of the root filesystem had better not be a `tar` archive or you will be in trouble. You should also ensure that any shared libraries needed by your desired utility are present. Be sure to verify this before the disaster occurs.

We will now look at this process on each of our Unix operating systems individually.

AIX: mksysb and savevg

AIX provides the `mksysb` utility for creating bootable backup tapes of the actual live system, which are self-restoring in the event of a failure. It saves all of the filesystems in the root volume group, generally */*, */usr*, */var*, */home* (unless you've moved it), and */tmp*, plus any paging spaces in *rootvg*. `mksysb` is invoked as follows:

```
# mksysb -i /dev/rmt0
```

`mksysb` relies on a data file that records various system configuration information. It is updated by including `mksysb`'s `-i` option. Use the `-m` option instead if you wish to restore the exact disk locations of the filesystems in the root volume group as well as their contents (`-m` says to save the logical volume maps as well as the other configuration information).

To restore the root volume group, boot from the `mksysb` tape and select the appropriate option from the resulting menu. The system will then be restored from the `mksysb` tape.

You can use a similar technique to clone a system from a `mksysb` tape made on a different system. If all the devices are identical, the only restriction is that you should not install a kernel from a multiprocessor system onto a single CPU system or vice versa.

When devices differ between the source and target system, a slightly modified technique is used. First, you boot off the install media, and then you select the option for restoring from a `mksysb` tape. In this mode, the operating system will automatically substitute drivers from the installation media when the ones on the `mksysb` tape are not correct for the target system. Note that this method will work only if the target system has the correct drives for accommodating both the `mksysb` and installation media simultaneously.

Restoring individual files from a mksysb tape

`mksysb` tapes can also serve as nonemergency backups of the root volume group. It is very easy to restore individual files from it. These tapes contain four distinct (tape) files, and the disk files from the root volume group are in the fourth file, which consists of a `restore` archive.

Thus, you could use the following command to restore the file */usr/bin/csh* and the subdirectory */etc/mf* from a `mksysb` backup tape:

```
# restore -s 4 -x -q -f /dev/rmt0 ./bin/csh ./etc/mf
```

The `-s` option indicates which tape file to use, and the `-q` option suppresses the initial prompt asking you to press the Enter key after you have mounted the first volume. Use `restore`'s `-T` option to list the contents of the archive.

Saving and restoring AIX user volume groups

The `savevg` command may be used to back up an entire user volume group, just as `mksysb` does for the root volume group. For example, the following command saves all of the files in the *chemvg* volume group to tape drive 1:

```
# savevg -i chemvg /dev/rmt1
```

The `-i` option creates the configuration file needed to save and restore the volume group; using `-m` instead also saves the logical volume maps, allowing their physical locations on disk to be reproduced.

savevg also has a -e option, which says to exclude the files and directories listed in the file */etc/exclude.vgname* from the save set.[*] Wildcards are not permitted in exclusion lists.

All of the logical volumes and filesystems and the files within them in a volume group can be restored from a savevg tape; the restvg utility performs this operation. For example, these commands restore the *chemvg* volume group we just saved:

```
# restvg -q -f /dev/rmt1
# restvg -q -s -f /dev/rmt1 hdisk4 hdisk5
```

The first command restores the volume group to its original disks, beginning immediately and without prompting for the first tape volume. The second command restores the structure and contents of the *chemvg* volume group to disks 4 and 5, shrinking all logical volumes to the minimum size necessary to hold the files within them (**-s**).

The tape made by savevg is a restore archive, so it is easy to extract individual files from it, as in this example:

```
# restore -f /dev/rmt1 -T -q
# restore -f /dev/rmt1 -x -q -d ./chem/src/h95
```

The first command lists the contents of the archive, and the second command restores the */chem/src/h95* subtree, creating any necessary subdirectories (-d).

FreeBSD

FreeBSD provides a several options for restoring system files, but all of them require that you have a complete backup of the filesystem from which to restore.

In the event of a system disk or boot failure, you must boot from alternate media (CD-ROM or a boot floppy). Then select the Fixit option from the main menu that appears. At this point, you can choose to boot from the second installation CD (which will function as a live filesystem) or a fixit floppy, or you can start a limited shell. The first two options tend to be the most useful.

The fixit floppy is a limited FreeBSD operating system containing enough tools to restore from a backup. It includes support for the tar and restore commands and tape devices. You create a fixit floppy by mounting the first installation CD and using a command like this one:

```
# dd if=/cdrom/floppies/fixit of=/dev/rfd0c bs=36b
```

This floppy can be customized after creation for your specific needs.

In order to save the disk partition layouts on a FreeBSD system, use the fdisk -s and disklabel commands. Along with */etc/fstab*, this information will allow you to reconstruct the disk partitions and filesystem layout. The disklabel command can also be used to write a boot block to a replacement system disk.

[*] The mksysb command also recognizes -e, and its exclusion file is */etc/exclude.rootvg*.

HP-UX: make_recovery

HP-UX provides the make_recovery facility for creating bootable recovery tapes as part of the Ignite-UX package (the utility is stored in */opt/ignite/bin*). A common method of using this utility is the following:

```
# make_recovery -p -A -d /dev/rmt/1mn
# emacs /var/opt/ignite/recovery/arch.include
# make_recovery -r -A -d /dev/rmt/1mn -C
```

First, we run the command in preview mode (-p). This command does not write any data to tape, but instead creates the file */var/opt/ignite/recovery/arch.include* which consists of a list of the items to be included. Here, we are choosing to save the entire root filesystem via -A; the default is to save only the subset of files that are part of the HP-UX operating system.

Once this command completes, we check the */var/opt/ignite/logs/makrec.log1* log file for any errors or warnings. If any are present, we must take any corrective action necessary and then rerun the first command.

Once any warnings are dealt with, the *arch.include* file can be edited to add or remove items, and then make_recovery can be run again in resume (-r) mode.[*] The -C option tells the command to update the stored data of the most recent make_recovery procedure.

This process must be repeated after each significant system change. The check_recovery command can be used to determine if make_recovery needs to be run.

Although these tapes are not intended to replace normal backups, it is possible to retrieve individual files from them. To do so, you must manually position the tape to the second file and then extract the desired items with tar:

```
# cd /
# mt -t /dev/rmt/1mn fsf 1
# tar xvf /dev/rmt/1m relative-pathname(s)
```

The file list should be specified as relative pathnames (e.g., *etc/hosts*, not */etc/hosts*).

> The most recent versions of the HP Ignite-UX package also provide make_tape_recovery (creates tape recovery images on the client system itself and from the Ignite-UX server) and make_net_recovery (write a recovery image to the disk drive of the Ignite-UX server across the network). See the documentation for details

[*] In some cases, additional considerations apply when some system files reside outside the root volume group; see the manual page for details.

Linux

On Linux systems, you can create a boot floppy of the current kernel with this command:

```
# dd if=/boot/file of=/dev/fd0
```

Simply copying the compressed kernel to diskette is all that is required, because the Linux kernel is structured so that it is the image of a bootable floppy disk (and it is loadable by either the DOS boot loader or `lilo`).

This procedure will enable you to boot your system should there be some problem booting from the hard disk. However, if your system disk is damaged and the root filesystem there is inaccessible, you will need a true recovery system to restore things. In such circumstances, you can boot using a *rescue disk*, which is created with the installation CD mounted with a command like this one:

```
# dd if=/cdrom/disks/rescue of=/dev/fd0 bs=18k
```

The rescue floppy contains tools needed to restore a saved backup, including tape devices and the `tar` command.

To record the disk partitioning information, use the `fdisk -l` command. Along with */etc/fstab*, this information will allow you to reconstruct the disk partitions and filesystem layout, and you can use `lilo` to create a boot block on a replacement system disk. Note that its `-r` option will prove very useful when the new root partition is mounted at some other point (e.g., */mnt*) within the rescue filesystem.

Recent versions of Red Hat Linux also provide a system rescue option when booting from the installation CD.

Solaris

Solaris provides little in the way of tools for system backup and recovery. You should make full backups of the root filesystem. You can then boot off alternate media to create a minimally working system and restore from your backup.

The `prtvtoc` command along with */etc/checklist* will provide the information required to recreate the disk partitioning and filesystem layout scheme. You can use the `installboot` command to write a boot block to the system disk. Note that boot images are stored within the installed filesystem at */usr/platform/model/lib/fs/ufs/bootblk*, where *model* is a string corresponding to your specific Sun hardware model (e.g., *SUNW-Sun-Blade-100*).

Tru64: btcreate

Tru64 provides the `btcreate` command for creating a bootable backup tape for the operating system. The tape consists of a bootable miniature operating system and a complete backup of the system files.

Running `btcreate` is very easy in that it will prompt you for all of the information that it requires. The default (suggested) answers are almost always correct. A restore from a `btcreate` tape will recreate the logical volume configuration from the original system in addition to restoring all of the system files.

On Tru64 systems, you can use the `disklabel -r` command to record disk partitioning information and recreate them if necessary.