Torture-testing Backup and Archive Programs: Things You Ought to Know But Probably Would Rather Not

Elizabeth D. Zwicky – SRI International

ABSTRACT

Many people use `tar`, `cpio`, or some variant to back up their filesystems. There are a certain number of problems with these programs documented in the manual pages, and there are others that people hear of on the street, or find out the hard way. Rumours abound as to what does and does not work, and what programs are best. I have gotten fed up, and set out to find Truth with only perl (and a number of helpers with different machines) to help me.

As everyone expects, there are many more problems than are discussed in the manual pages. The rest of the results are startling. For instance, on Suns running SunOS 4.1, the manual pages for both `tar` and `cpio` claim bugs that the programs don't actually have any more. Other ''known'' bugs in these programs are also mysteriously missing. On the other hand, new and exciting bugs – bugs with symptoms like confusions between file contents and their names – appear in interesting places.

The Tests

The test suite currently employs two sorts of test. First, there are static tests; files and directories with stressful names, contents, or permissions, which do not change while the program runs. Second, there are active tests; files that change while the program is running.

Static tests:

1. A file with a large hole in it.
2. A file that contains a hole and a block's worth of nulls.
3. Files with funny characters in their file names.
4. 1025 hard links to the same file.
5. 2911 hard links to different files.
6. Files with long names.
7. Symbolic links to long names.
8. Symbolic links to names with funny characters in them.
9. Unreadable and unwriteable files.
10. Unreadable and unwriteable directories with normal files in them.
11. A named pipe.
12. A device.

Active tests:

1. A file that becomes a directory.
2. A directory that becomes a file.
3. A file that is deleted.
4. A file that is created.
5. A file that shrinks.
6. Two files that grow at different rates.

Some errors occur between multiple backups or during multiple restores, and the test suite does not test for them at this point. I will mention these conditions further later.

The tests were run using a perl program which created all the static files, and then forked to modify files in one process and run the program being tested on the other. All programs were tested through a pipe, rather than having them actually create and read from tape archives. There were no compatibility tests; the archives were always being read by the program that wrote them (or, in the case of `dump`, by `restore`.) Except where specified, they were run with no options beyond those required to archive to standard output and read from standard input. A modified version of GNU diff was used to compare the original directory with the restored one.

Files With Holes In Them

Many versions of UNIX are capable of storing files with large numbers of nulls in them using what are called ''holes''. If a program seeks over empty portions of a file, instead of explicitly writing them, the operating system may choose to avoid actually allocating disk space for those portions. This substitution is mostly invisible to programs. Programs that read the data in the file will not be able to tell which nulls are written on the disk, and which are not. In some versions of UNIX, including POSIX-compliant versions, `stat` will return the number of blocks actually used and the size of a block; comparing this to the length of a file allows a program to determine how many holes there are, but not where they are. In other versions, not even this information is available to a program making normal use of the file system.

Programs like `dump` that read raw disk devices have no difficulty with holes, since they are reading the blocks directly. Programs like `tar` and `cpio` that read files through the file system are almost incapable of getting holes right. There are three possible algorithms for a completely portable program to use for holes; it can ignore them, and always write nulls; it can assume that anything that could be a hole is, and never write nulls; or it can attempt to determine where the holes are by rewriting the file with explicit nulls and seeing if the file gets any longer. The last option has obvious flaws (not only is it painfully slow, but it requires a writable file and at least one block's worth of free space on the file system), and as far as I know has never been implemented[1].

Always filling in holes is the worst of the two plausible options; the most frequent fatal flaw occurs on core files in /. Core files are the most common files with holes in them, and the core files in / often have enough holes to make them the size of the virtual memory in the machine. Since / itself is often smaller than the virtual memory of the machine, a backup of / that fills in the holes may be unrestorable.

Always creating holes is somewhat better. Some programs use blocks full of nulls as a cheap way of reserving space on a disk for future use; the most common example of this these days is `mkfile` on SunOS 4.0 and above, which is used to preallocate swap files for NFS-based swapping. Replacing the nulls in a such a file with holes will have a truly unfortunate effect on performance on the machine using them for swap. It will be completely fatal if the space freed by the holes is then used by other files, since the holes cannot be filled when the machine needs the space. On the other hand, programs that do this are quite rare, and creating holes is less fatal than filling them.

Programs that are willing to sacrifice portability by requiring a `stat` that returns block size and number of blocks can always get the correct number of holes. They can't guarantee that the holes will be in the right places, and there are theoretical situations where this would be problematic – for instance, a program that intermixed seeks where it knew it would not need to put real data in, and writes of nulls to reserve space, working on a full disk, would require that the holes be where it left them[2]. I have never actually run across such a program; then again, I am not aware of any programs using this algorithm, either.

The test cases are a file which has a nearly 10 megabyte hole in it, and a file which has a block full of nulls and then a hole. The latter case tests for programs which always create holes.

[1] Barry Shein invented it solely to disprove my claim that a truly portable program working through the file system could not possibly get holes right.

[2] This example comes from Dave Curry.

Funny Characters in File Names

There are persistent rumours that some versions of `tar` are incapable of backing up files with control characters in their names. This appears to be false; on the other hand, it is quite true that newlines in file names upset `find`, which is often used in conjunction with `cpio`. It is also true that some `tars` are incapable of writing files with funny characters in their names, even when running on operating systems that allow them.

I tested characters from octal 001 to octal 377, where possible (some operating systems do not allow characters with the high bit set). I did not test ''/'', because my test suite runs at too high a level to create them. ''/'' is completely illegal in UNIX file names, in all versions of UNIX, because it is reserved to the kernel for use as a separator. Unfortunately, it is not impossible to create files with ''/'' in their names. Most NFS implementations access the filesystem directly, without going through the normal kernel routines, and few of them error-check names. NFS clients running on machines like Macintoshes, where ''/'' is a legitimate character, may be capable of creating these files. (Sites using `aufs` to provide file service to Macintoshes are safe, since it runs as a normal UNIX process. Furthermore, on directories that it considers Macintosh-native, it performs transparent translations between ''/'' and '':'' since '':'' is a reserved separator on the Macintosh.)

By definition, no program running within the UNIX filesystem can deal with files that have ''/'' in their names; `dump`, like NFS, runs below the filesystem, but `restore` does not. Versions of `restore` that allow you to restore files by inode number will allow last-ditch retrieval of these files. None of the other programs tested could possibly have allowed the backup of these files, which is why I was not particularly concerned about missing that test.

Even between UNIX machines, there are file name problems. A machine running HP/UX, for instance, may be running with a 14-character file name limit. Not only will this cause problems with archives written on machines with longer limits, but it may cause problems writing archives on that machine, if it NFS mounts file systems from other machines or provides NFS service to other machines. Preferably, archive programs should avoid implementing the file system limits of the machines that they are running on while they are writing archives.

In order to achieve maximum nastiness, and also to create a very large number of files, each character appears alone as a file name, as the beginning of a file name, as the end of a file name, and as the middle of a directory name. Each directory has 10 plain files in it. For each character, the program also makes a symbolic link with the octal value as the name, and the character as the target. This test actually turned up a nice feature in GNU tar; in verbose mode, it prints the C codes for funny characters, resulting in a much more readable listing than any other program provided.

Large Numbers of Hard Links

In the ''BUGS'' section of the cpio manual page, it states ''If there are too many unique linked files, cpio runs out of memory and linking information is lost thereafter.'' The question is ''How many is too many?'' The test program creates a hard link to every file in the set with funny names (this is the primary reason for the 10 files in each oddly-named directory). It also creates 1024 extra links to a plain file, just to check for programs that have problems with large numbers of links to the same file.

Long File Names and Symbolic Link Targets

The tar manual page documents a 100-character path length limit; the cpio manual page under SunOS 4.1 cpio documents a 128-character path length limit. The pax manual page more vaguely admits ''there are restrictions on the length of pathnames stored in the archive'' because of restrictions on the formats it uses. In tar's case the 100-character restriction is indeed a firm upper limit because of the format; in cpio's it is not, and varies by implementation. (Most versions of tar do not make it all the way to the 100-character limit, so there is minor room for variation by implementation there, too – there are also people intentionally running mutant tar-like programs that produce incompatible tapes with a longer limit.)

100 characters may be a reasonable limit on systems with a 14-character file name limit, where you have to type all the characters in every file name yourself. On systems where file names may go up to 255 characters, and filename-completing or icon-based shells are available, users may overrun 100 character limits quite frequently.

There are two constraints on name length; a limit on how long an individual component of the path may be, and a limit on the total length of the path. On BSD-based UNIX implementations, these are usually 255 and 1023 characters, respectively. What many people fail to take into account is that these are both per-filesystem limits; your 1023 character pathlength limit starts counting at the beginning of the name of the mount point, not at /. Simply allocating a 1024 character buffer – or even finding the maximum path length and allocating a buffer that size – does not guarantee you will actually be able to fit any file name, starting from /. Nothing I've found, including dump and restore, actually manages to archive a file with a name that is the maximum allowable. To be fair, under SunOS 4.1, fsck considers a file with a name over 1021 characters to be an error, also.

Many archive formats silently implement a limit on the length of the target of a symbolic link, either by failing to write the link at all, or by truncating it. Some will notify you that they were unable to archive the link. So far, I haven't found a program with a limit that also correctly documents what the limit is.

Unreadable and Unwriteable Files and Directories

An archive program that is being run by root on a local disk should have no problems with file permissions. Users running their own archives, or root running over NFS, may run into permission problems. Some permission difficulties are well-known – for instance, the cpio manual page recommends running find depth-first so that files will appear on the tape before their containing directories, in order to avoid difficulties with directory permissions.

These tests are run as the user who owns the files; theoretically, the archive program could have forcibly read the unreadable files by changing the permissions. Whether this would be a good thing or not is an open question.

Named Pipes and Devices

Some archive programs, like tar, archive only ''normal'' files. Others will pick up special files as well. The most spectacular failures here, under Encore Mach, are not actually the fault of the archiver; that version of the operating system turns out to crash whenever you do almost anything to a named pipe, including remove it.

A program that does not archive special files is going to be a severe annoyance if you need to restore an entire system, since you will have to build at least /dev by hand.

Hard Links to Directories

I ran some tests with hard links between directories in place. It is not a standard part of the test suite, because it wreaks even worse havoc than the normal tests; it becomes very hard to determine exactly what is causing things to fail. Furthermore, the number of non-archive programs that fail becomes a real trial. All the archive programs fail, one way or another, including dump. Hard linked directories are an error, and will be caught and removed by fsck; on the other hand, very few people run fsck before every backup. In most cases, the hardlinked directories are skipped. Systems using find ended up skipping the hardlinked directories, but achieved this effect by being massively confused, providing dozens of confusing and irrelevant error messages (mostly complaining that the files in the hardlinked directories didn't exist). tar actually succeeds in backing them up – the hardlinks disappear in the restored version.

Active Tests

The active tests are not as complete as the passive ones, for several reasons. First, I wasn't able to run the active tests on all of the machines that the passive tests were run on; the passive tests could be run on any machine that could NFS mount a directory, but the active test required a functioning perl. Perl is probably capable of running on all the machines, but I wasn't in a position to install it on all of them in time to run the tests. Second, many of the active tests only produce errors if the timing of the events happens to be just right. Third, there are cases in which programs provably get active tests wrong, but silently compensate for the error in most cases. I intend to work up a more effective test suite (and to install perl on the machines that currently don't have it). Meanwhile, I will sketch the problems tested for.

Programs that do not go through the file system, like `dump`, write out the directory structure of a file system and the contents of files separately. A file that becomes a directory or a directory that becomes a file will create nasty problems, since the content of the inode is not what it is supposed to be. Restoring the backup will create a file with the original type and the new contents.

Similarly, if the directory information is written out and then the contents of the files, a file that is deleted during the run will still appear on the tape, with indeterminate contents, depending on whether or not the blocks were also re-used during the run

All of the above cases are particular problems for `dump` and its relatives; programs that go through the file system are less sensitive to them. On the other hand, files that shrink or grow while a backup is running are more severe problems for `tar`, and other file system based programs. `dump` will write the blocks it intends to, regardless of what has happened to the file; if the file has been shortened by a block or more, this will add garbage to the end of it, and if it has lengthened, it will truncate it. These are annoying but non-fatal occurences. Programs that go through the file system, on the other hand, write a file header, which includes the length, and then the data. Unless the programmer has thought to compare the original length with the amount of data written, these may disagree. Reading the resulting archive – particularly attempting to read individual files – may have unfortunate results[3].

Theoretically, programs in this situation will either truncate or pad the data to the correct length. Many of them will notify you that the length has changed, as well. Unfortunately, many programs do not actually do truncation or padding; some programs even provide the notification anyway. In many cases, the side reading the archive will compensate, making this hard to catch. SunOS 4.1 `tar`, for instance, will warn you that a file has changed size, and will read an archive with a changed size in it without complaints. Only the fact that the test program, which runs until the archiver exits, got ahead of `tar`, which was reading until the file ended, demonstrated the problem. (Eventually the disk filled up, breaking the deadlock.)

Untested Problems

The test suite in its current states looks only at problems that arise within a single archive. There are other problems that arise when you use programs designed for single archives to do multi-level dumps, attempting to pick up files that have changed since the last pass.

Programs that work through the filesystem (including `tar`, `pax`, and `cpio`) modify the inode access time. This is also the only time that is changed when changes are made to the permissions on files, so systems using these programs cannot also pick up changes to permissions. (This is the same problem that `rdist` exhibits.)

Programs that are not designed for multiple-level backups (again, including `tar`, `pax`, and `cpio`) mark only existing files on their archives. Since they don't know about past state, they cannot mark files that have been deleted or renamed since previous runs. This is not a problem as long as you are making backups, but if you need to restore them, the result is likely to be more files than space; every file that ever got onto a backup will be restored, and every renamed file or directory will be present twice, once under each name.

Other Warnings

Most of the things that people told me were problems with specific programs weren't; on the other hand, several people (including me) confidently predicted correct behavior in cases where it didn't happen. Most of this was due to people assuming that all versions of a program were identical, but the name of a program isn't a very good predictor of its behaviour. Beware of statements about what ''tar'' does, since most of them are either statements about what it ought to do, or what some particular version of it once did. Also watch out for people who state that they've never had a problem backing up files of some type; what you care about is whether you can back them up and then restore them. You can back up files with ''/'' in them, for instance, until the cows come home, without problems. You just can't restore them.

Don't trust programs to tell you when they get things wrong either. Many of the cases in which things disappeared, got renamed, or ended up linked to fascinating places involved no error messages at

[3] "cpio out of phase: get help!" springs to mind

all.

Some programs had peculiarities which were not consistent, or weren't relevant to the tests. pax, for instance, worked in some situations only if run in verbose mode. Under Encore Mach, tar core dumped while running on the funny names on one run (complaining multiple times of an unknown write error 70, and finally saying ''HELP'' and dying). This behaviour did not reappear on later runs. It also tended to produce error messages in blocks, with first the content part of 20-40 messages, and then the ''tar:'' from the beginning of all of them. Not all programs got multiple test runs, so in some cases I may have hit or missed intermittent problems.

Although the test suite runs lots of tests, it is by no means exhaustive. For instance, since I set the suite up I have heard it suggested that some versions of cpio convert symbolic links to hard links where possible. The test suite does not create any valid symbolic links, so this problem would not show up. There are also problems that are extremely rare – for instance, when a dump tape begins with the continuation of an inode which is another dump.

The testing directories tend to turn up bugs in unexpected places; anything that tries to walk directory trees is probably history. That includes not only find, but also du, rm -r, diff, and even gnudiff. Painful experience taught me that while I was working in the testing directories, I needed to turn off the cd alias that showed the current directory in the prompt. Not only do prompts longer than a few hundred characters cause the version of tcsh I run to core dump, but the alias I use won't let me out of a directory with a space in its name. I also learned to delete the test directories promptly, to avoid the complaints of my co-administrators about the mail generated by dying cron jobs, and the censure I got for having a directory that du found an extra 300 megabytes in.

Conclusions

These results are in most cases stunningly appalling. dump comes out ahead, which is no great surprise. The fact that it fails the name length tests is a nasty surprise, since theoretically it doesn't care what the full name of a file is; on the other hand, it fails late enough that it does not seem to be an immediate problem. Everything else fails in some crucial area. For copying portions of file systems, afio appears to be about as good as it gets, if you have long file names. If you know that all of the files will fit within the path limitations, GNU tar is probably better, since it handles large numbers of links and permission problems better.

Looking at tables in Appendix A, it's easy to fall into a deep depression (after the initial incredulity wears off). It's worth remembering that most people who use these programs don't encounter these problems.

My testing programs are available for anonymous ftp from ftp.erg.sri.com, with some warnings. Chief among them is the warning not to run them on a machine that's critical; even if the tests don't crash it, and they probably won't, you'll immediately be blamed for anything that goes wrong.

Author Information

Elizabeth Zwicky is a system administrator for the Information, Telecommunications, and Automation Division at SRI International, where she relieves tension by torturing innocent file systems. Reach her via U.S. Mail at SRI International; 333 Ravenswood Avenue; Menlo Park, CA 94025. Reach her electronically at zwicky@erg.sri.com.

Appendix A: Tables of Evaluations

	Large Hole	Deceptive Hole
Tar (all known versions except Gnu)	Filled in	Filled in
Gnutar 1.10 with -S	Correct	New hole created
Cpio (all known versions)	Filled in	Filled in
Pax (SunOS 4.1)	Correct	New hole created
Afio (SunOS 4.1)	1 block of nulls filled in	Correct
Dump (all known versions)	Correct	Correct

Table 1: Holes in Files

	File Names	Symbolic Link Targets
tar (SunOS 4.1, HP-UX 7.0, Irix 3.3.2)	Correct	Correct
tar (Encore Mach 1.0, Mt Xinu Mach)	Files with 8th bit set missing	Correct
pax -x ustar (SunOS 4.1)	Correct	Correct
Gnutar 1.10	Correct	Correct
find | cpio (SunOS 4.1, HP-UX 7.0)	Files with newline missing (problem is in find)	Correct
find | cpio (Encore Mach 1.0)	Files with newline or 8th bit set missing	8th bit stripped
find | cpio (Mt Xinu Mach)	Files with newline or 8th bit set missing	Correct
pax -x paxcpio (SunOS 4.1)	Correct	Correct
find | afio (SunOS 4.1)	Files with newline missing (problem is in find). Some files became executable; some directories became normal files. The problem does not appear to be funny characters, since the affected directories included "151(i)dir" and others that had only normal characters in the name.	7 symbolic links converted to regular files. Their original targets were "!", "#", "K", "L", "N", "O", and "T".[4]
find2perl -cpio[5]	Correct	Correct

Table 2: Funny Characters in Names

[4] This doesn't seem to be a funny character problem, but I have no idea what it is.
[5] find2perl: 2-line modification to quote file names.

	Unique links	Multiple links
tar (SunOS 4.1, Encore Mach 1.0, HP-UX 7.0, Irix 3.3.2)	Correct	Correct
Pax -x ustar (SunOS 4.1)	256	Correct
Gnutar 1.10	Correct	Correct
find | cpio (SunOS 4.1)	All linked files are still linked, but only 2268 of them correctly - the remainder are crosslinked among each other, with link counts up to 8	Gained 2 extras out of the unique links
find | cpio (Encore Mach 1.0, Mt. Xinu Mach, HP-UX 7.0)	Correct	Correct
pax -x paxcpio (SunOS 4.1)	Two showed up linked to the multiples; rest were not linked at all	The right number, but 2 of the wrong files, 2 of the originals omitted
find | afio	One linked to multiples; some not linked at all; some of the linked ones apparently linked to the wrong places; some of the linked ones added execute permission.	Right number of links, but one of them to the wrong place, and the last one of the originals omitted
dump (SunOS 4.1)	Correct	Correct

Table 3: Large Numbers of Hard Links

	File becomes directory	Directory becomes file	File is created
Tar (SunOS 4.1)	Directory	File	File exists
Gnu tar 1.10	Directory	File	File doesn't exist
afio	Directory	File, but with 2 links	File exists
find2perl -cpio	Directory	File	File exists

	File is deleted	File shrinks	File grows
Tar (SunOS 4.1)	File doesn't exist	Apparently OK	Writes until end-of-file
Gnu tar 1.10	File doesn't exist	Notes that file has shrunk, but claims it has shrunk by 0 bytes	OK
afio	File doesn't exist	OK	OK
find2perl -cpio	File doesn't exist	cpio became out of sync and exited	Untested

Table 4: Active Tests

	Pathname length	Symbolic link length
tar (SunOS 4.1, Encore Mach 1.0, Mt. Xinu Mach, Irix 3.3.2)	99 characters (100 claimed in man page)	98 characters
tar O (HP-UX 7.0)	99 characters	98 characters (100 claimed in man page)
tar N (HP-UX 7.0)	125 character pathname (100 character filename) (256 claimed in man page)	98 characters (100 claimed in man page)
Gnutar 1.10	86 characters; overlength files are put in ./MaNgLeD<filenumber>, with warning	99 characters; overlength silently truncated
pax -x ustar (SunOS 4.1)	99 characters; at 100 writes the file but appends ''000644'' to the end of the name. Does not produce an error message until much later. In directories above the pathname limit, puts files on the tape as ./filename, truncating the filename at 100 characters and apppending 000644	99 characters; above 100 characters appends ''ustar'' to the end of the link but writes it anyway without comment.
find | cpio (SunOS 4.1)	Writes all the files, but gets confused at 258 characters and complains that it cannot change mode on a file that has the first 258 characters of the path name, plus the contents of the file, as its name (128 claimed in man page)	Correct (tested to 256)
find | cpio (Encore Mach 1.0, Mt Xinu Mach)	Writes all the files, but gets confused at 256 characters and complains that it cannot change mode on a file that has the first 256 characters of the path name, plus the contents of the file, as its name (128 claimed in man page)	Correct
pax -x paxcpio	253 characters, complains about a damaged archive	255 characters
find | afio (SunOS 4.1)	Find complained about pathname too long on the long directory; using find2perl, afio succeeded	Correct
dump (SunOS 4.1)	1021 characters (OS limit is 1023; the difference appears to be dump appending ''./'' to the pathname); restore gives up even earlier, at 1015 characters	Correct

Table 5: Long File Names and Long Link Names

	Unreadable file	Unreadable directory	Unwriteable file	Unwriteable directory
tar (SunOS 4.1)	Missing	Exists, with contents, is now drwxr-xr-x	Correct	Exists, with contents, is now u+w
tar (Encore Mach 1.0)	Missing	Exists, new permissions, no contents	Missing	Exists, with contents, is now u+w
tar (Mt Xinu Mach)	Missing	Exists, is now drwxr-xr-x	Correct	Exists, with contents, is now u+w
tar (HP-UX 7.0)	Missing	Exists, new permissions, no contents	Correct	Exists, with contents, is now u+w
tar (Irix 3.3.2)	Missing	Exists, no contents	Correct	Exists, no contents
pax -x ustar (SunOS 4.1)	Missing	Causes the reading pax to core dump	Correct	No contents
Gnutar 1.10	Missing	Exists, wrong permissions[6], no contents	Exists, wrong permissions[6]	Exists, with contents, wrong permissions[6]
cpio (SunOS 4.1, Mt Xinu Mach)	Missing	Exists, correct permissions, no contents	Correct	Correct
pax -x paxcpio (SunOS 4.1)	Missing	Causes the reading pax to core dump	Correct	No contents
find | afio (SunOS 4.1)	Missing	Exists, correct permissions, no contents	Correct	No contents
dump (SunOS 4.1)	Correct	Correct	Correct	Correct

Table 6: Difficult Permissions on Files and Directories

	Named Pipes	Devices
Tar (SunOS 4.1)	Missing	Missing
Tar (Encore Mach 1.0)	Machine crashes	Missing
Tar (Mt. Xinu Mach)	Correct[7]	Correct
Tar O (HP-UX 7.0)	Missing	Missing
Tar N (HP-UX 7.0)	Correct	Correct
Pax -x ustar (SunOS 4.1)	Correct	Correct
Gnutar 1.10	Correct	Correct
Cpio (SunOS 4.1, HP-UX 7.0)	Correct	Correct
Cpio (Encore Mach 1.0)	Machine crashes	Correct
pax -x paxcpio (SunOS 4.1)	Correct	Correct
Afio	Became regular file	

Table 7: Special Files

[6]Gnutar warned that it was changing the permissions; the only change was the application of the current umask.

[7]The OS does not support named pipes, so creation fails, but the correct data is present in the archive and tar attempts the creations