REPOSITORY MIRRORING

ERIC GROSSE*

Abstract. Distributed administration of network repositories demands a low overhead procedure for cooperating repositories around the world to ensure they hold identical contents. *Netlib* has adopted some refinements on the widespread scheme of anonymous `ftp` and `ls-lR`. Checksum files and a two small C programs give an easily maintained system that copes with communication breakdowns and subtle changes in repository contents. The packaging of these C programs inside a shell pipeline provides an explicit command stream that can readily be checked before execution. Protecting files, keeping logs, and so forth becomes effortless and reliable. The same tools, applied on a smaller scale, allow more people to participate in the editorial work of maintaining a high-quality repository, by eliminating the need for directly manipulating files at remote sites.

1. Introduction. The most striking change in publication practices in scientific computing over the past decade has been the phenomenal growth in network repositories of software, test data, benchmark results, preprints, and other products of research and administration. Increased network speeds and ubiquitous connections have been a key factor in this growth. At the same time, a trend away from "computer center" to "personal workstation" computing has increased the desire for fast access to individual functions on demand, rather than prospective installation of commercial libraries by centralized staff. As algorithms improve and more sophisticated modeling problems are attempted, the need grows for specialized functions beyond the scope of traditional libraries. All of these trends are expected to continue.

Together with correspondence by email, access of these archives represents the main use of networks today in the numerical community, though one can forsee distributed computation and real-time collaborative links growing in importance. The improved scope and peak quality (though perhaps not the average quality!) of these archives has led to sufficiently heavy use that new problems of administration are emerging.

Traditionally, repositories have been administered at one central site. To the extent that one repository borrows from another, the data flow tends to be haphazard, leading to the presence of obsolete copies of software on the net. As it becomes feasible to have joint efforts that span many organizational and geographic boundaries, a more distributed yet coordinated approach becomes necessary. The best candidate for an editor of a subject area like linear regression may be an expert statistician knowledgeable in the analysis of algorithms, rather than someone with broad computing systems experience. (By "editor," we mean the same sort of person who manages a technical journal.) Such an editor is likely to have a much easier time editing files on a local file system than learning how to use ftp to download files from the central repository, edit them, and then reload the repository. Even for an expert, this process is tedious and prone to mistakes in the presence of interruptions and network outages.

The solution adopted in `netlib`[1], the dominant source of freely available mathematical software, is to allow each editor to work at the most convenient site that can provide file access, by ftp or other means, to the main netlib servers. Netlib takes responsibility for keeping each of its directories synchronized with the master copy at the editors' sites. By the same means, each of the cooperating repositories can stay synchronized. The mirroring procedure that achieves these goals is the subject of this note. §2 describes the checksum file generated nightly at each site and used by each slave to look for changes in the master. §3 shows how this information is used to actually bring the sites into agreement. §4 compares this approach with previous work.

Scaling of the repository system to accommodate growth in usage and to make more efficient use of trans-oceanic links and the like forces the replication of repository servers at

* AT&T Bell Laboratories, Murray Hill NJ 07974 USA. `ehg@research.att.com`.

distant sites. Keeping the repository contents at each site synchronized is another instance of the same problem as tracking editors' changes.

The mirroring system described here is now in daily use, making sure the various netlib servers around the world hold the same contents. Contributions from specific areas, such as the typesetting and membership files from SIAM, are automatically updated in netlib by the same mechanism. We also use the tools as a partial solution to the problem of synchronizing files on a network of workstations and laptops[2]. Based on this experience, the tools (themselves available from netlib, of course) appear suitable for other groups with similar distributed administration needs.

2. The central convention, a checksum file. Each master site for a collection of files maintains a listing of these files that permits slaves to quickly identify changes. The format of this listing is one line per regular file, containing four fields separated by white space:
name pathname relative to the top-level directory;
mtime modification time, in seconds since 00:00:00 GMT, Jan. 1, 1970;
length number of bytes in file;
crc a 32-bit CRC hex checksum.
For example, the first three lines of the checksum file for `research.att.com` are

```
a/beatson 495383361 21424 c2b189b1
a/blur/cart.uu 720909866 1384347 155dae94
a/blur/index 720909941 241 b56f07b5
```

The file is sorted and saved in compressed form.

The checksum was implemented by James W. Williams of the University of Maryland[1] and is freely available by

```
mail netlib@research.att.com
send crc.c from crc.
```

The checksum was based on a draft of the POSIX 1003.2 standard[6], but unfortunately the final version of the standard changed the definition slightly; it didn't seem worth the effort to switch simultaneously at all netlib sites, unless there were some significant advance like a demonstrated need for a cryptographically strong checksum. Input to `crc` is a list of files, typically created by the program `lsr.c` (written for this mirroring project and also available from the same directory in netlib.) This file list may be filtered to exclude administrative or secret files that should not be mirrored.

Each repository maintains a top-level `crc` directory, containing checksums of the files for which this repository is the master, as well as checksums of files that it gets from other repositories. Editors and repository users can then readily check whether the latest versions of files have been installed everywhere. These checksums in `crc` are constructed by an automatic process running just on the local disk, and hence are quite reliable compared to anything involving network access.

Why use a checksum at all? There are a several reasons that have arisen in recent netlib experience. Experimental networks may be temporarily untrustworthy. A file on the slave may have the wrong time, but the right contents because of transfer through another channel. Much worse, a file on the slave might have the right time but the wrong contents because of bungled manual attempts to keep things in sync.

If file contents are always copied via `tar` files (which carry along accurate time information), the netlib experience is that checksums can as a practical matter be dispensed with except for occasional (say, annual) verifications. Indeed, an earlier mirror scheme used in

[1] This product includes software developed by the University of California, Berkeley and its contributors. The particular copy in netlib also contains a standalone program to print a file in the format described above.

netlib took this approach. However, we found that the need for running tar commands on both master and slave sites and finding space for the tar work file (for large additions like a new release of `lapack`) made this approach impossible to fully automate.

Because the format of the checksum file will be hard to change once it is in wide use, some care must be taken at the beginning to select a sensible choice. In the interests of making this an open system and encouraging alternative implementations in other contexts, the emphasis was on easy parsing, to wit

```
while(scanf("%2000s %lu %lu %lx",name,&tim,&len,&sum)==4)
        process(name);
```

Contrast this with the messy code needed to parse the so-called `ls-lR` files (recursive Unix[2] directory listings) that are traditionally used in mirroring.

An argument against the proposed format is that it wastes space because many of the bytes on one line are repeated on the next. But experimentation shows that, after compression, this disadvantage vanishes. For the 5174 netlib files with master location at `research.att.com`, the checksum file is 205748 bytes uncompressed, 92671 compressed. Converting the integers to base94 notation (since that can be represented using ordinary printing ASCII characters) and eliding pathname prefixes that are unchanged from one line to the next reduces the size of the uncompressed checksum file to 114795 bytes, but the compressed size only goes down to 80435. Compression can't do much on the checksums, so dropping those from the file (retaining just the filename, date, and length) makes more of an improvement: uncompressed 83509, compressed 51313. For comparison, the same information in the form of `ls-lR` (which contains no checksums) is 337224 uncompressed, 68359 compressed. Experiments with other lists of files gave similar results. The conclusion is that the checksum format is easier to parse than `ls-lR` and contains more information in not much more space.

3. The `masterslave` **and** `mirror` **commands.** Now, suppose we have two checksum files from a master repository and a slave. The term "master" is relative to a specific list of directories; each of the netlib sites at present is master for some of its contents and slave for other parts. The arrangement is symmetrical and has no designated most important node. Besides the technical aspects, this symmetry has psychological advantages in the context of administration distributed across organizations.

A small program, `masterslave`, compares the two checksum files and prints a command sequence that updates the slave. The mutually exclusive commands are

get copy a file from master to slave;
rm remove a file from slave;
time update mtime of a file in slave that already has contents identical to master.

These commands, after being checked for plausibility, are fed to another program, `mirror`, that executes the `rm` and `time` commands immediately, makes sure that directories and files mentioned in `get` commands are ready for writing, and creates an `ftp` script. Any other file transfer mechanism could equally well be used, but `ftp` is widespread and has a fairly standardized interface. We also use `ftp` to acquire the checksum files before running `mirror`. After transferring the files, a final check step verifies that the new checksum matches the expected one and, if all is well, sets the modification time of the file to the expected value.

Of these three commands, `time` is relatively unusual; other mirroring tools are properly suspicious of using file size alone and retrieve a fresh copy whenever dates do not agree. Netlib uses `time` more often than one might expect, because some of the smaller editorial sites find it expedient to copy all files nightly to an ftp server. This is undesirable, but something we

[2] Unix is a registered trademark of UNIX System Laboratories

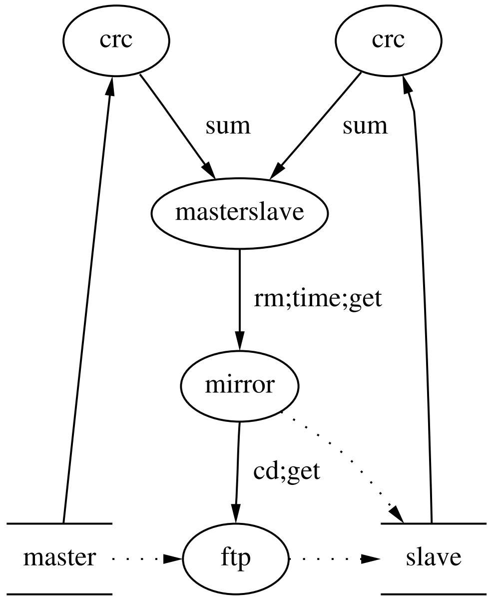

FIG. 1. *Master and slave repositories are synchronized by comparing checksums of files, removing files on the slave, setting times on the slave, and copying files by ftp from the master.*

can tolerate temporarily in order to promote distributed administration. Checksums make it efficient and safe.

Note that files are read directly to their final location and no significant temporary file space is needed. Also, note that no special permissions are needed, except perhaps to set up the nightly checksum process.

We turn now to various refinements.

If checksums are expensive, it is better to recompute a file's sum only when its date or length change. The `masterslave` program, fed by `lsr` and the old sum file, can easily implement this incremental strategy. When checking dates for this purpose, one ought to use something like the Unix `ctime`, which is difficult to forge and therefore tends to accurately reflect the last time the file was touched in any way, rather than the Unix `mtime`, which is commonly used in netlib to indicate the release date of the code as chosen by the author. (Other operating systems may use different names for these concepts, but the point stands.) It is wise to compute a checksum file from scratch periodically, to catch disk errors or other subtle effects.

As a matter of netlib policy with the interests of users topmost in mind, file names are supposed to use only ordinary printing characters, not be too long, not involve symbolic or hard links, and generally speaking be portable to many systems. We have not gone quite as far as requiring that the file names meet MS-DOS's severe limits, though we do try to avoid case sensitivity to accommodate people from older systems. At some point, when ftp clients are believed to be capable of dealing with such generality, we hope to allow UTF[4] filenames, and will introduce an escape convention in the checksum file names.

Quis custodiet ipsos custodes? Given the state of the art in wide area networks, aborted file transfers in ftp must be considered. If an ordinary file is lost, no great harm is done; the file will be resent at the next mirroring session. If the checksum file itself is only partly transferred, however, a naive slave might remove large parts of its collection, thinking that the master had just done so. Especially in the presence of ftp gateways, we are unwilling to depend on failure notification from ftp.

Hence, as mentioned earlier, we check the proposed set of changes for plausibility before executing them. If more than, say, 50 files or 2 megabytes would be removed by the script, it is not automatically executed but instead a message is sent to the site administrator for manual intervention. The master also provides, for those slaves who wish to be extra cautious, a checksum file of the master checksum files. The latter is not a replacement for the former; we wish to protect against a disaster on the master infecting the entire world overnight. Disasters on netlib so far have been rare, but as the number of master sites and the range of expertise broadens it has to be expected that files will accidentally be removed through mechanical or human error.

If the checksum file contents are not public, encrypt and continue to use anonymous ftp. A secure file transfer protocol could be used if preferred, but this seems an unnecessary complication for implementers and an inconvenience for users.

A more serious security issue is that the slave is implicitly allowing the master to control its file system, with the obvious risk of tampering with password files and so forth. Our plausibility checker protects against this by watching for attempts to write files outside the archive.

For ease of administration it is convenient to have the archive files in their natural form, but to make responsible use of network resources, all file transfers should use compression. Complicating the issue is the fact that some files are already compressed in their natural form. To meet these needs, the `mirror` program reads a table of suffix rules to decide for each file transfer whether to use compression.

Source for all these programs is also available from the `crc` directory of netlib.

4. Comparison with other approaches. My initial preference was to make use of an authenticated, wide area, portable, network filesystem. A number of possibilities were explored, none of which looked fruitful on a near-term time scale. As operating systems research continues and migrates to standards and products, this situation ought to improve.

The best existing package, as far as I know, that addresses the same task is the ftp mirroring software, `ftp-mirror`, by Lee McLoughlin.[3] The package consists of several large Perl scripts using `ls -lR` directory listings. That approach has the obvious and very valuable advantage of being a more generally applicable solution, since it is usable with ftp sites that are not willing to run `lsr` nightly. The `ls -lR` convention, while not universal, is fairly common. In the worst case, the slave can issue repeated `dir` commands to laboriously build the file listing itself. The inherent disadvantage of `ls -lR` is the lack of checksums and the imprecise dates (given to nearest minute or day, and with no indication of timezone).

Besides the differences in assumptions, the `netlib` strategy uses 700 lines of C in place of 5000 lines of Perl in `ftp-mirror`. Not all the netlib sites have yet installed Perl, though Larry Wall's outstanding `Configure` script should make the installation task merely a matter of finding disk space. Operating performance is dominated by the actual file transfer, so is similar for both implementations.

Reasons for building a new system included:

1. In the most compelling case seen so far in netlib, a TOMS algorithm acquired a subtle bug by having an assignment statement commented out, leaving the file size unchanged. Much to our consternation, the bug reappeared twice (quickly spotted), as revised versions of the algorithm were submitted by people who had captured

[3] Available by ftp from `src.doc.ic.ac.uk` in `/packages/mirror/`, with communications software by A. Macpherson, Alan Martello, and Randal Schwartz.

an intermediate buggy version. Because one bug suggests the possible presence of others, we obtained a fresh copy of the file from the original author. The "modification time" of the file changed non-monotonically during this period, as we tried to keep it an accurate reflection of the author's true modification date. Only a system based on checksums can hope to recover from episodes like this.

2. The `ftp-mirror` package does its file checking while connected to ftp on the remote system, hence needing code for keep-alive messages and, worse, tying up a possibly scarce connection on the remote system. When master and slave elect to store files in different compression schemes, even comparing sizes is a nontrivial operation.
3. Modern ftp servers often supply either compressed or uncompressed versions of a file. The `ftp-mirror` package would see only the uncompressed file in the remote `ls-lR` listing and spend more network bandwidth than necessary.
4. The `ftp-mirror` package has to go through some contortions to deal with the interaction of compressed and split files, particularly upon deletions. The `netlib` package recommends that the master archive contain files of reasonable size in the first place, though no fixed limit is built in. Once again, this reflects the conscious trade-off that makes the `ftp-mirror` package broadly applicable and the `netlib` package more efficient for loosely cooperating sites.
5. Although automatic mirroring does most of the installations in netlib, it sometimes happens that a file is added first at a slave repository, and email sent to another editor asking for installation at the master. Somewhat more often, a file is sent by its author to several repositories and installed at each. In this situation, it is convenient for the slave to just update the file time from the master and not recopy the file. Checksums allow this (minor) optimization.

A system like `ftp-mirror` or `netlib` puts most of the burden on the slave to decide what to do in order to mirror files from the master. Urban Mueller had the clever idea to do just the opposite for `aminet`.[4] A single master site takes responsibility for creating ftp scripts for each slave; the slave just does as it is told and reports when done. This reduces to a bare minimum the effort to set up new mirror sites, though it does violate the symmetry principle mentioned earlier and requires an element of trust.

Some mirroring systems use the `ftpget` command by Mark Moraes.[5] This is a C program that directly connects to an ftp server and transfers files, a facility that would save about 100 lines out of the `netlib` package. (`ftpget` also contains an option for identifying files that are out of date, but this suffers the same problems described above.) Unfortunately, my site has had to adopt the "firewall" approach to security and depends on a `proxy-ftp` program put in place by a system administrator. In the Unix tool tradition, `netlib` emits a command file for the established `ftp` interface.

A convenient wide-area-file-system is Dave Presotto's `ftpfs` in the Plan9 operating system[5]. This allows one to make a remote file system, accessible by ordinary ftp, appear to be in the local file space. (Similar functionality is available for Unix systems from packages such as Prospero.) Installing `ftpfs` or something like it in each mirror location would be a burden because some of the netlib repositories are running very old versions of Unix. However, I do use `ftpfs` when bypassing the automatic system and manually making changes or comparing files.

Although unable to use these other packages directly, I studied them for ideas and wish to acknowledge their contribution.

---

[4] Relevant files are `pub/aminet/priv/site.doc` and `poll.sh` at `wuarchive.wustl.edu`.

[5] Available from `nigel.msen.com` in `pub/packages/networking`.

Acknowledgements. I thank Ron Boisvert, Shirley Browne, Bill Coughran, and several anonymous referees for improving the presentation, Dave Presotto and Phil Winterbottom for sage advice on the problem, Ken Thompson for accommodating my desire to set mtime in Plan9, and Steve North for the *dot* system that drew Figure 1.

REFERENCES

[1] J. J. DONGARRA AND E. GROSSE, *Distribution of mathematical software via electronic mail*, Comm. ACM, 30 (1987), pp. 403–407.

[2] J. H. HOWARD, *Using reconciliation to share files between occasionally connected computers*, in Fourth Workshop on Workstation Operating Systems, IEEE Computer Society, 1993, pp. 56–60. Napa, California.

[3] JUVENAL, *Satires*. (Who is to guard the guards themselves?) p.347 of Lewis Evans translations, according to Oxford Dictionary of Quotations.

[4] R. PIKE AND K. THOMPSON, *Hello world or* $\kappa\alpha\lambda\eta\mu\epsilon\rho\alpha\ \kappa o\sigma\mu\epsilon$ *or . . .*, in USENIX Winter 1993 Conference Proceedings, 1993.

[5] D. PRESOTTO AND P. WINTERBOTTOM, *The organization of networks in Plan 9*, in Winter 1993 USENIX Conference Proceedings, 1993, pp. 271–280.

[6] TECHNICAL COMMITTEE ON OPERATING SYSTEMS, *Portable Operating System Interface, Part 2*, IEEE Computer Society, 1983.