The OSU-CIS Backup and Restore System

Steve Romig

March 15, 1994

Contents

1 Introduction **2**

2 Overview of the Backup System **4**
- 2.1 Tapes 4
- 2.2 Database 5
- 2.3 Programs 6
- 2.4 Configuration Files 7
- 2.5 Features 7

3 Installation **9**
- 3.1 Things to do once... 10
- 3.2 If you are creating a new database... 12
- 3.3 If you already have a database... 14
- 3.4 Things to do on each host... 15

4 Testing **17**

5 Configuration **18**
- 5.1 global.defs, local.defs 19
- 5.2 backup.config 19
  - 5.2.1 chains 19
  - 5.2.2 level 20
  - 5.2.3 location 21
  - 5.2.4 offsite 21
  - 5.2.5 group 21
  - 5.2.6 tape-type 22
  - 5.2.7 tape 22
  - 5.2.8 config-name 22
  - 5.2.9 Miscellanea 23
- 5.3 $db_dir/backups 23

6 Doing Backups **24**
6.1 Adding Tapes to the Tape Database 24
6.2 Starting a Backup Run 25
6.3 Checking a Backup Run 26

7 Doing Restores **27**

8 Tape Management **28**

9 The Graph Making Stuff **28**

10 Daily Procedures **32**

11 Disaster Recovery **32**

12 Problems that Crop Up **33**

13 Archive Programs **35**
13.1 Adding Support for New Archive Programs 36

14 Future Plans **37**
14.1 Multiple Tape Support 37
14.2 Control of Incrementals 39
14.3 Online Catalogue of Files 40
14.4 Backup of Active File Systems 40
14.5 Interoperability 40

A The Backup Database **41**
A.1 Tape Database 41
A.2 The Run Database 42

1 Introduction

This document describes the The Ohio State University Computer and Information Science (OSU-CIS) Department's backup and restore system. The system still doesn't have a name so I'll just refer to it here as "the backup system". This document describes version 2.6 of the backup system.

The backup system consists of a set of Perl scripts (and 1 small C program) that provide what is intended to be a comprehensive and configurable set of tools for doing incremental backups and restores. The main design constraints were that it should be easy to use, robust in handling errors, complementary to human skills and shortcomings, and that it should use native backup programs.

"Easy to use" means that the programs that are used most, `all-backups` and `getback`, are user friendly in that they convey a consistent and simple interface to handling backups on many and diverse systems. If you fail to provide necessary information on the command line, they'll ask for what they want. An on-line database is used to make tape management and restores simple.

"Robust" means that the system tries really, really hard to deal with errors. Programs that might damage or change the tapes or the online database check to ensure that the right tapes are being used in the right drives. There are all sorts of internal consistency checks intended to find bugs and problems in the database. It ain't perfect, but its better than a crufty set of shell scripts with little or no error checking.

"Complementary to human skills" means that it tries to make up for what people are bad at, and tries to avoid what people are good at. It seems hard to write a program that could schedule backups and correctly take into account tape drive use, drive status, and operator schedules, so that is left to the users (if you have ideas about how to automatically schedule backups, assign tape drives and so on, please tell me about them!) People are forgetful and sometimes forget to do backups, and its easy to tell when one is overdue, so there's a `check-backups` program that will complain about overdue backups.

"Uses native backup programs" means that it should be easy to use these tools with just about any sort of archiving program and media. There are a plethora of file archiving programs and file system types. You've got short and long name file systems, BSD Fast File Systems, file systems that support strange objects like context dependent files or access control lists, and strange proprietary database file systems that have their own weird backup programs. You probably already have archiving programs (variants of `dump`, `tar` or `cpio`) that can "correctly" save and restore these things. What you are probably missing is a framework that handles tape management and error handling and all that stuff. This backup system supplies that framework, allowing you to use the most appropriate archiving program for each file system without rewriting the rest of the backup system.

This version supports incremental backups in multiple chains on multiple hosts, tape types, fairly easy restores, backups to tape or disk or both, parallel backups, event logs, offsite tape storage, and even more error checking than the previous versions.

No, it still doesn't allow backups to span multiple tapes. Sorry. See section 14 for details about this and other future plans.

This manual describes the backup system (section 2), how to install it (section 3), how to do backups (section 6) and how to do restores (section 7). I'm aiming this document at describing the overall system and the philosophy behind it. More detailed information about the various programs is available in their man pages.

2 Overview of the Backup System

The backup system consists of backup, restore, and database management software. The programs that are intended for human use are `all-backups`, `getback`, `add-tape`, `move-tape`, `show` and `reuse`. In addition, `run-clean-db`, `offsite`, `check-backups`, `backup-report` and `list-backups` are utility programs that are usually set up to be run by `cron`. Everything else is used by those programs in some fashion or another.

2.1 Tapes

Let's start by talking about tapes. Each tape to be used by the backup system is registered in the tape library through the `add-tape` program. `Add-tape` writes a tape label at the start of the tape that includes a unique tape identification name. This name is used to identify tapes so it should be written onto the physical tape label in some fashion - I suggest writing the tape ID in ink. The tape label also contains information about what the tape was most recently used for so that if the physical tape label falls off you should still be able to identify the tape by reading the first file (it is a text file written in a simple format). The `add-tape` program also enters the tape into the tape library database so that it can be used for backups.

Normally all you will need to do to maintain the tape database and tape library is to add new tapes as required and replace tapes that have worn out. The `add-tape` program can be used to add new tapes and to relabel old tapes whose tape labels are bad. The `reuse` program is used to tell the tape management system that a particular tape can be reused right away rather than waiting for it to be recycled when the backups on it expire. This is mostly used when a backup run fails and you want the tape to be available for other backups.

Each tape has a tape ID. The ID consists of a prefix of some sort and a number, such as "exb-10", "rack1-12", or "test-tk50-123". The prefix defaults to the type of the tape, but can be any arbitrary string (you can use the `-p` option to `add-tape` to set the prefix for a tape that you are adding.) The number is used to make the IDs with the same prefix unique. The prefix defaults to the tape type so that you can store tapes of the same type in a rack in sequential order (otherwise tapes 1 through 9 might be 8mm, but tape 10 is a tk50 and 11 is a 9 track reel - how would you rack them?) The system allows you to change the prefix so that you can make up your own racking system.

Each tape also has a location and a home. The home is where the tape is supposed to reside when it isn't offsite and the location is where the tape currently is. If you have tapes that were added with pre-2.6 software, the location and home fields for those tapes might not be set. The location and home are mainly used for offsite tape storage and to ensure that `all-backups` will use tapes from the free tape pool in the same location as the tape drive.

2.2 Database

The backup system maintains an on-line database that covers the tape library and all of the current backups. The databases are implemented as `dbm` files accessed as associative arrays from within Perl. If you are backing up more than one host, you can share the database through NFS. Unfortunately the `dbm` library that is normally used with Perl is byte order dependent and will not work correctly if you try to share the database files between hosts that don't use the same byte ordering. The BSD `db` library provides a `ndbm` library interface but is written to support byte order independent files that can be shared through NFS. All you need to do to use it is to construct a version of Perl that uses that library instead of the usual `dbm` or `ndbm` library and convert your existing database files from `dbm` format to the new format. There are a few things you'll need to change in the configuration files - see the installation instructions (section 3) for details on dealing with this. If you can't use NFS, then it should be possible to rewrite the database module (all the database routines are in one file) to allow access through some network connection or something like that. Contact me for more information - some people have started working on something like that already.

More information about the internal structure of the databases can be found in appendix A.

The on-line database records information about:

- each tape, how old it is, how often it's been used, what it is currently being used for, what its type and current format is, what drive it was last written in.
- each backup run (a collection of one or more backups that were made through a single invocation of `all-backups`). What level of backup this is, what backup chain this is, which hosts were included in the run, when the run was started and when it finished.
- each backup. What run it was part of, what file system on which host, what span of time that the backup covers, what tape and which file on that tape, where it can be found online (if at all), what the status is, and what type of archive it is (e.g. `dump` or GNU `tar`).

The database information and tape labels are used all over the place to make things easy to use and relatively safe from accidental mistakes. For example the tape labels are checked prior to using the tape to make sure that the correct tape has been inserted in the tape drive to keep you from accidently overwriting the wrong tape. The backup information is used to make the process of restoring files easier by finding the appropriate tapes for the task for you.

The databases are maintained automatically through the `run-clean-db` program that is (supposed to be) run nightly by `cron` from one (1) host. The `run-clean-db` program expires old backups and runs from the backup and run databases. Once it knows which backups and runs are still current it frees any tapes that no longer contain current backups so that they can be reused for new backup runs.

2.3 Programs

Finally we come to the backup and restore software. Backups are initiated with the `all-backups` script. `All-backups` can be used to start backups on one host or many hosts. Like most of the rest of the software it is set up to ask you questions about what you want to do if you don't supply the necessary information on the command line, so it should be pretty self-explanatory to use. The collection of backups initiated through one invocation of `all-backups` constitutes a backup run and is identified by a unique run ID number. The `show` program can be used to view the contents of the databases associated with a particular run or to view the entries in the tape library.

The restore program is called `getback`. `Getback` is intended to be very easy to use. You tell it what host and file system you want to restore the files from and it tells you what backups to use and walks you through the process of extracting the files. It supports three "modes": interactive, full and pick. In interactive and full restore modes `getback` selects what it thinks is the best sequence of backups to use to restore files from and prompts you to insert the corresponding tapes, one by one, starting with the lowest level and working toward higher levels. The only difference between the two is that full mode extracts everything on the backup, and interactive mode allows you to use `restore i` to selectively add and extract files from each backup (this only works with backups made with `dump`). Pick mode shows you a list of all of the backups for the host and file system that you are interested in, and allows you to select which backups to extract files from.

The `check-backups` script can be used to check to see what backups are due to be done on a particular host. I suggest setting it up to run from `crontab` nightly on each of the hosts that is to be backup up where it can serve as a reminder of missing backups for those who are supposed to do backups (and their supervisors :-).

The `list-backups` program prints a summary of all of the backups contained in the database. You can print this list out periodically and use it to do restores if the machine that hosts the backup database is down or if the backup database is damaged.

If you store some of your tapes offsite, you can use the `offsite` program to generate a report for each offsite storage location that lists the tapes that should be in storage there, what tapes should be checked out to that site (and then taken there), and what tapes should be brought back and check in. The `move-tape` program is used to update tape locations as you move them to offsite storage and bring them back again.

2.4 Configuration Files

You're probably wondering how this whole thing is configured and controlled. There are four sources of configuration information:

- Per-host installation information in the `global.defs` and `local.defs` files, which controls the locations and general operations of the programs.
- The global `backup.config` file in the database directory, which controls overall backup operations (what levels, how often they should be done, etc.)
- The `$db_dir/backups` file which controls which file systems get backed up, for which levels, and how.
- The command line on the `all-backups` command, which determines which hosts will be included in each backup run, what chain and level to use, what tape drive, and how the run should be executed.

See the section on configuration (section 5) for more details.

2.5 Features

Centralized Control This system makes it easy to initiate backups on multiple hosts in a consistent fashion while allowing for individual variations among the hosts. "Consistent fashion" means that the same chains and levels are run on the various hosts with the same frequency, using a common tape library. "Individual variation" means that its still easy to determine which file systems should be backed up, what should be used to back them up, and what levels should be allowed for each chain for that file system.

Easy To Configure The system is controlled by 3 global configuration files (`global.defs`, `backups` and `backup.config`), and by one host-specific file (`local.defs`).

Tape Aliases You can define convenient aliases for tape host and device specifications, which makes it more convenient for both backups and restores.

Tape Type Support The system can deal with information about tape types and formats to make it easier to store your tapes neatly and to correctly ascertain that it is using the right kind of drive for a given tape.

Host Groups You can also give groups of hosts a name, which makes it easy to start a backup run on all of them.

Tape Management The system deals with tape allocation and recycling automatically, and will expire tapes as they become too old or overused. You just need to add more tapes as required.

Online Record of Backups The backup system maintains an online database that records what each tape has been used for, what backup runs have been made, which hosts were involved in each run, what file systems on those hosts were backed up, when the backup finished, what the state of the backup was, what errors occurred, and what tape and file number the backup can be found on (and/or disk file). This is used wherever possible to make it easy to find and restore from backups.

Error Checking Each tape has a tape label at the start that is used wherever necessary to ensure that you are using the correct tape in the correct tape drive. This helps prevent accidental overwriting of tapes with `add-tape` or `all-backups`, or incorrect restores through `getback`.

Backup Checker The system provides a program that can be used to automatically warn you when a backup is due on any of the file systems that are supposed to be backed up.

User Friendly The most commonly used tools will ask you for needed information if you don't supply it on the command line. All of the prompts give help if you type "?". The restore system helps you select backups to restore from and skips to the correct file on the tapes after you load them.

Multiple Independent Backup Chains This system provides the ability to make several independent backup chains, to help avert the problems caused by missing or damaged backup tapes.

Multiple Archive Programs This system currently supports the use of both GNU `tar` and BSD `dump`, and it is fairly easy to add support for other programs. This means that you should be able to coordinate backups of special file systems (such as database partitions) along with the backups of your normal file systems on the same tapes. If you are really paranoid about backups, you can also do parallel chains with different archive programs to effect more complete independence among your backups.

Flexible File System Configurations You can choose what chains and levels apply to each file system, and what archive program to use. This means that you can do only chain 0 full saves on one file system using `dump`, and all chain 1 backups on another file system using GNU `tar`.

Concurrent Backups You can do multiple backups in parallel on many hosts and have the results copied sequentially onto the tape as they are completed to save time.

Backup During Boot You can schedule backups to be done automatically while the file systems are quiescent during the early part of the boot process.

Reschedulable Backups You can schedule backups to start at a later time. This means that at the end of your working day you could set up a full save to run at 5:00 A.M. the next morning in single user mode, and sleep through the whole thing.

Incremental Safety Latches The system tries to detect situations where a full save should be done instead of an incremental, and warn you about them, for example if a full restore was done more recently than the latest full backup, or if it thinks a lower level backup has never been done.

Fine Control Over Backup Runs You can use "tags" to control whether to include or exclude certain file systems from a backup run, so that you can split the file systems of several hosts onto separate tapes to ensure that they will fit.

Online Backups You can do backups to tape, disk or both. If a backup is found online somewhere, the restore facilities will use that instead of the tape copy.

Multiple Configs on Same Hosts You can use the `config-name` entry in the `backups.config` file to give names to separate incarnations of the backup system, so that you can run a production version of the system (config-name set to "") and a test version (config-name set to "test") at the same time without interference between them.

Online Log File The tools in the backup system maintain a online log of important events. Entries are added when tapes are added to the database, when backup runs are started, and when restores are done.

Offsite Tape Storage Management The backup system provides support for offsite tape storage. A simple program tells you what tapes to take offsite and to bring back. The backup and restore programs take a tapes location into account to avoid using tapes that are offsite unless you specify otherwise.

Cute Graphs If you use BSD `dump` for your backups you can use the graph making stuff to make interesting graphs that display run sizes and durations, backup sizes for each file system, tape use and so on. This is somewhat useful for planning - you can find out how much data you are putting on your level 2 backup tapes, for example.

3 Installation

This system is intended to be used in an environment where multiple hosts share a common database of tape and backup information. So there's one database that has to be set up in such a way that it will be available to all of the hosts in question. And each host needs its own copy of the programs and scripts. I suggest that you read carefully through all of these instructions before starting to follow them.

3.1 Things to do once...

The following things should only need to be done once.

1. CD to the `src` directory.

2. If you have RCS, you might want to make sure that everything is checked out of the RCS subdirectory by running `co RCS/*`.

3. Edit the `global.defs` file. This defines the names and locations of the backup database and other global information. The things that you will need to change are:

   $our_timezone the local time zone.

   $default_tape_type the default type for tapes.

   $default_tape_format the default format for tapes.

   $backup_dir this is the default location for on-line backups.

   online_repositories This is a list of directories where online backups are kept. This is used in `run-clean-db` so that it can find them and delete any expired backups when it cleans house. This should include `$backup_dir` and any other online backup directories that you use.

   $db_dir where the database files are.

   $db_server the name of the host where the database files are really located.

   $db_ext this should be the extension that your `dbm` routines puts on the `dbm` files. If you are using `dbm` or `ndbm`, this should be "dir", if you are using the BSD `db` library use "db".

   $db_ext2 this should be the other extension that your `dbm` routines puts on the `dbm` files. If you are using `dbm` or `ndbm`, this should be "pag", if you are using the BSD `db` library use "".

   $db_server, $db_remote, $db_local are the host name, export path and local mount point for the file system that contains the backup database. These are used within `single-begin` and `single-end` to mount the file system so that the programs can access the database when doing backups in single user mode.

   $bdir where the backup programs are (same as BDIR in the Makefile).

You shouldn't need to change anything else, but I do recommend that you read through the file and see whether the default settings are what you want. You should also check the names of the database files in `global.defs` to see if they match your current names. If you are running on a short name file system (where file names are limited to 14 characters or less), the default database names will probably cause problems for you. You might be able to work around the limitations by making the database names shorter. Note that in the case of the per-host run and backup databases, the names of the hosts will be added to each file name. Note also that I have changed the names from the values used in release 2.5, so if you haven't upgraded since then, you'll either need to change the values in `global.defs` or rename your database files to match the new names.

4. Take a look at the `local.defs` file also. I believe that I have it set up in such a way now that you can use the same `local.defs` file on just about any version of UNIX. You may need to make minor adjustments for different types of hosts. If this is a System V Release 4 host then set `$svr4` to 1. Check the `@fs_types` array to see if your systems file system types are listed (these are the "type" values that you find in the `/etc/fstab` file. You will need to change `$bitch_prog` to something suitable. This is executed when the tape checker finds that the wrong tape is in the drive. You should read through the rest of the file and make sure that the defaults apply to your host. If you are using GNU `tar` version 1.10 or greater you shouldn't have to change anything, otherwise you will have to adjust the various GNU tar flags. If the local version of `dump` can handle remote device specs like `user@host:device` then change `$dump_does_remote_users` to 1. If you are using Ultrix but you are NOT using the default dump that comes with the Ultrix distribution, you need to change `$ultrix_dump` to `0`. If you don't have GNU tar installed then comment out the definition for `$gtar_prog`.

5. Set up `clean-db` to run nightly on the host that holds the backup database. It should be run on the backup database since it needs to have guaranteed access to the database files if it runs. This should be set up to run at a time when it is unlikely that any backups will be run to avoid database contention since it locks several databases, possibly for a longish time. There shouldn't be any problems if it is run while a backup is being done however. You might want to use the `-v` flag in addition to the `-y` flag and pipe the output into `mail` to send it to a list of interested parties. This can produce lots of output when run on a large database, however. There's a program named `run-clean-db` that you can use to run `clean-db` once a day, logging the output to a subdirectory of `$db_dir`, and sending all the interesting bits on through mail. We run it like this:

```
0 1 * * * /usr/local/utils/nbu/run-clean-db
```

6. Set up `list-backups` to run nightly on some host. `list-backups` prints a report showing the known backups for each file system. The idea is that you print this daily and keep it in a log book for reference in an emergency when the online backup database isn't available. Since the report can get rather large I suggest that you print the whole thing once a week and then print incremental updates once a day. We do something like the following:

   ```
   0 6 * * 0  list-backups | lpr
   0 6 * * 1  list-backups -N 1 | lpr
   0 6 * * 2  list-backups -N 2 | lpr
   0 6 * * 3  list-backups -N 3 | lpr
   0 6 * * 4  list-backups -N 4 | lpr
   0 6 * * 5  list-backups -N 5 | lpr
   0 6 * * 6  list-backups -N 6 | lpr
   ```

   which prints the whole thing on Sunday, then prints 1 days worth on Monday, 2 days worth on Tuesday, etc. We keep Sunday's and today's listings in the notebook and throw the rest away. If you use off site tape storage, you should make a copy of the `list-backups` reports and store them at each offsite tape storage location for use in disaster recovery.

7. You can use `backup-report` to print a table that shows the latest backups in table form, where you can easily recognize "gaps" in the backup coverage and take appropriate actions. We run it once a day as:

   ```
   30 6 * * * backup-report -N 14 | lpr
   ```

8. If you are doing offsite tape storage, then you should set up the `offsite` program to run daily on some host (probably the same one that runs `run-clean-db`). The `offsite` program generates a report that lists the offsite storage locations, the tapes that should be stored at each, a list of tapes to take to each location, and a list of tapes to be brought back from each one. You could set up the `crontab` entry to mail the report to interested parties, as in:

   ```
   0 14 * * *  offsite | mail -s "Offsite Storage Report"\
           romig fine
   ```

3.2 If you are creating a new database...

If you are creating a new database directory for this installation then please follow the following instructions.

1. Create the \$db_dir (the database directory) on a suitable server. You'll also need to create \$clean_db_log and \$error_dir (these are subdirectories of \$db_dir by default). The backup programs run as user root, group backup. The database files and directories are set to be owned by group backup, mode 770 (660 for files) so that the programs can still write them but so other users can't read them. Change \$db_dir, \$clean_db_dir and \$error_dir to be owned by group BACKUP (or whatever), mode 770.

   You could set things up to allow read access to OTHER but if I ever add the ability to maintain an on-line list of what files are in which backups then such open access to the database might be inappropriate, since making them world readable would allow everyone to see the names of everyone else's files.

   Note that the \$db_dir must be available on each of the hosts that will use this backup system, under the name \$db_dir. You could do this through NFS, for instance. If the database directory cannot have the same name on all hosts then you will have to adjust your `global.defs` file on a host-by-host basis.

   The hosts that access the database directory through some sort of network file system will be running programs as root, group backup. These programs must be able to write the database files, and you may need to work some magic to get that to work right. If you are using NFS to share your database directory, the server that holds the database must meet one of these requirements:

   (a) It has a clean implementation of NFS that allows programs to access files according to group write permissions even when running as root. SunOS 4.1 has a bug wherein root cannot access files even if it is running with the group that has specific write access to the file. To test, create a directory on the server in question owned by group backup, mode 770, and a file within that directory, group backup, mode 660. Make a setgid version of cat, group backup on some client that mounts that file system. Run that cat with the pathname of the file you just made. If it works, you're set. If it doesn't you'll have to either find a server that works or revert to method b.

   (b) It allows root access for all of the hosts that will be backed up. You can do that either by setting a `root=server1:server2:server3` line in your exports file (if supported) or by adding an entry like "anon=0" to the appropriate entry in your `/etc/exports` file.

2. Copy `samples/backup.config` to \$db_dir and edit it. Refer to the man page on `backup.config` for details. See the section on configuration (section 5) for more details and ideas.

3. Copy `samples/exclude` to $db_dir. This lists files that should be excluded from the backups. The distributed exclude file keeps the system from backing up files named buid-NNN. This is actually used only by the GNU `tar` backup code so if you don't plan to use that you don't need the exclude file.

3.3 If you already have a database...

If you have an existing database directory and you are just updating the software then there are still some things that you need to do.

1. The entries in the `backup.config` file have changed somewhat. These are the changes in version 2.5:
   - The `level` entries now mention the chain number.
   - There are `tape-type` entries, and the `expire-uses` and `expire-age` entries have gone away.
   - The `tape` definitions now include the type, read and write formats.
   - There's an optional `config-name` entry that you can use to set up a test version of the software.

   In version 2.6, the following changes were added:
   - There's a `location` entry which must be used to declare each tape drive/storage location.
   - The `tape` entries now include `status` ("yes" if "mt status" reports the correct file and block number, "no" otherwise) and `location` (the location of this tape drive).
   - The `default-tape` entries are now specific for particular tape types and formats.
2. If you are running version 2.4 software or earlier, then you will need to use the 2.5 database conversion programs (contact Steve Romig, romig@cis.ohio-state.edu, for details). If you are running version 2.5 software, you don't have to do much. I've changed the names of the sequence files to match the corresponding backup database files - this will be handled automatically by `run-clean-db`, though you will have to delete the old `-id` files by hand. I've also changed the names of the database files in `global.defs` to eliminate the superfluous ".db" - you can either change the names in `global.defs` to match the old names, or rename your database files to match the new names.

I suggest that you make a copy of your existing database, install the new software, edit `global.defs` and `local.defs`, rename your backup database files, delete the old `-id` files, and run `clean-db -v` to create the new sequence files. If there are any mismatches between the real file names and the names in `global.defs`, it should become painfully apparent, and you can interrupt `clean-db` and correct the problem. Repeat until everything seems to work smoothly.

3.4 Things to do on each host...

1. Install the man pages somewhere.

2. Edit the Makefile - you might want to change the definitions of PERL, SDIR (where the Perl scripts go), BDIR (where the binaries go), PRINTER (where you print your sources) and the "make print" entry itself. Oh, you'll need to update the BACKUP variable also if you decide to change the name of the "backup" group.

3. Create SDIR and BDIR. I'd suggest making them the same for simplicity, its up to you. You probably want a separate copy on each server that you are backing up so you don't have to depend on some other server being up to do backups on the others.

4. Create the backup group using whatever name you want. The value of the GID doesn't matter - we use 31.

5. Type `make` to compile the C programs (this only needs to be done once for each group of machines with the same binary format and OS release) and then `make install` to install the binaries, scripts and symlinks as needed. The Makefile uses `maketime.` to try to avoid doing unnecessary work. This may get in your way - if so just delete it before running `make install`.

   If you don't have `yagrip.pl` installed where Perl can find it you'll need to take the copy from the `misc` subdirectory and either install it in your Perl library directory or in SDIR.

6. Create a `backups` file that lists the file systems to be backed up. See the backups(5) man page for details. In previous versions, this would have been `/etc/backups`, but the default now is to use `$db_dir/backups` so that there is one place from which you can control what file systems are backed up and so on. If you were using `/etc/backups` you might want to pull them all together into `$db_dir/backups` (don't forget to prepend the proper hostname to each line). A minimal `backups` file might look like this:

   ```
   host:/          dump
   host:/usr       dump
   host:/home      dump
   ```

which means "do all backups (all chains, all levels) on those three file systems when this host is backed up, and use `dump` to do all of them". See the section on configuration for more suggestions (section 5).

7. If you are going to try to do backups in single user mode you may need to modify `single-begin` and `single-end` to suite your particular setup. `single-begin` should set things up so that the backup database is available (e.g. mount its NFS file system or whatever), and `single-end` just unsets everything so that the rest of the boot process can proceed normally. You will also need to modify the `/etc/rc` file to look for and run the `/backup` file if it exists. Here's a code fragment that will do the trick:

```
if [ -r /backup ]; then
        (echo 'Backing everything up...') >/dev/console
        mv /backup /backup.delete
        /backup.delete                    >/dev/console 2>&1
        rm -f /backup.delete
fi
```

   This should be inserted just after it becomes possible to mount NFS file systems (so you can get to the backup database) but before you start the NFS mount daemon (so that your file systems will be quiescent when the backups are run, which is the whole point of this exercise).

   Wait! Don't save that file yet. The backup commands in `/backup` will save their output to `/tmp/backup-output`. If it weren't for the fact that we're in single user mode we'd try to mail it. If you want the output mailed to you add this code somewhere near the end of your rc file (after the `sendmail` daemon is started):

```
if [ -f /tmp/backup-output ]; then
        (echo 'mailing backup results...') >/dev/console
        /usr/lib/sendmail -t < /tmp/backup-output \
                >/dev/console 2>\&1
fi
```

8. If you want to use this to backup hosts with local disks that are also part of an HP cluster, then you will need to create a CDF (Context Dependent File) named /backup so that when you try to do single user backups on several hosts at once they won't have nasty name conflicts. Of course, you won't be doing a single user backup of the server at the same time (since the whole cluster will be down), but you might want to do single user backups of several clients at the same time (we have some clients in clusters which have local disks that need to be backed up, in other words, diskless HP clients with local disks.)

9. Set up `check-backups` to run by `cron` nightly on each backup host. I would suggest using `check-backups -m -n`, which will cause it to mail the results to the addresses defined in the `backup.config` file and to send mail ONLY if there's a problem.

10. Edit the `.rhosts` files. Each of the hosts that will be backed up must trust the host from which the master `all-backups` program is started. This is because `all-backups` uses `rsh` to start do-backups on each host in turn. In addition all of the backup hosts must be in the `.rhosts` file on each of the tape hosts since GNU `tar` and `dump` use `rsh` as root to allow for network access to the tape drive. You can test your root rsh access from each host by running something like `rsh host hostname` from a host pointed at some other host. For example, run this from the host where you will start the backup runs from pointed at each of the hosts to be backed up and then from each of those to each of the tape drive hosts.

    If your version of `dump` and `GNU tar` support it, you can specify a remote user in the tape definitions in `backup.config` and tell the system to use that user by changing `$gtar_does_remote_users` and `$dump_does_remote_users` to 1.

    You can try to get around the need to have the backup hosts trust root on the host you start the backups from. Try changing `all-backups` to run `do-backups` with an `rsh host -l user ...'`. If that user has execute access to the backup programs, and the wrapper for them is running setgid to group backup, then that part of things should work. If you are using `dump`, make sure that group backup can read the disk devices and edit the dumpdates files, or you might try making a separate copy of the wrapper that is setuid root, setgid backup, and point the `backup-dump` and `backup-gtar` links in `$bdir` at it so that the backups will run as root.

4 Testing

Your backup and restore system is one of the most critical systems in your installation. If you've ever lost a disk or file system without a good backup to restore files from, you can appreciate the importance of a backup system in a personal way. If you've never lost a file system, never had a disk crash, never lost days or weeks (or years) of work on a computer with no way to restore the missing data, then believe me, you want to avoid the experience.

I have added some limited regression tests for some of the low level routines in the system. There's a test bed for testing the database routines, the file locking system, the "get_recent_backups" routine and various other pieces of the software. My goal is to add regression test routines that can be run more or less automatically after changing the code to help ensure that future changes don't break the system.

If you make significant changes to the internals of the system, you can do a sort of general regression testing by copying your current database and comparing the results of running the old software on it against the results of the new software. For example, I recently did some extensive testing by copying our (large) database to two test directories. I set up 2 copies of the backup software, one with the old software pointing at one database, the other with the new software pointing at the other database directory. Then I ran the same programs (at the same times) in the 2 installations and compared the results. Try `list-backups`, `show tape` and `show run`, and `getback`. When you are fairly certain that the software is working right, try `clean-db` (which might be destructive), and then run through the others again to make sure that `clean-db` didn't damage your database.

Part of the problem with debugging a backup system is that certain events can take a long time. For example, checking to see that a backup expires after 7 days, or that `check-backups` complains about the right things after several days of skipped backups, or that the `offsite` program moves tapes back and forth correctly. One partial solution is to change the length of a "day" from 24 hours to 30 minutes or so. You can do this by adjusting the value of `$DAYS` in `global.defs`. That way you can cycle through 2 "days" per hour of real time, and determine whether the various programs that use days as the basic unit of time are working correctly.

I've also tried to reduce the pain associated with testing backup systems by making it possible to install several copies of the backup system on the same hosts and have them cooperate without interfering with each other so that you can use a production version of this system while testing a new release. See section 5.2.8 for more details.

Finally, I've created a null backup type that goes through the motions of making a backup, without actually doing a backup. This is useful in testing the infrastructure. Just use type "null" in your `$db_dir/backups` file instead of `dump` or `gtar`.

5 Configuration

Some of this information is repeated in the installation instructions (section 3), but I'll include a brief description of everything here just to be complete. There are several sources of configuration information in this backup system: the `global.defs` and `local.defs` files, the (single) `backup.config` file, and the `$db_dir/backups` file.

Once you've set things up, you should think long and hard before changing anything in the config files. Some things are safe to change at will. Some things are dangerous. I've tried to make the difference clear in the comments.

For example, if you want to move the backup database, make the appropriate changes to `$db_dir` in `global.defs` and move the directory. You can add host groups, tape types and aliases, and change the list of mail recipients in `backup.config` without major repurcussions. Changing the level definitions may significantly change the daily operations and general utility of the backup system. Changing the configuration name after you have already set up the database is a definite no-no.

5.1 global.defs, local.defs

The `global.defs` and `local.defs` file contains installation information that isn't likely to change once you've installed the system and gotten it working. Some of the information may actually change from host to host, but most should be the same. The installation instructions cover setting this file up in some detail, so I won't rehash that here. Suffice it to say that this is where the directories that contain the binaries, Perl scripts and databases are named, among other things.

The comments in these files should indicate what's safe to change and what's not so safe.

5.2 backup.config

The `backup.config` file is more interesting. It should be installed in the database directory, and it contains global definitions that affect all of the hosts that use that database directory. These definitions affect more of the operational concerns of the system: what backups to do, how often to do them, how long to keep them before reusing the tapes, when to expire tapes, and who to send reports to. It can also be used to give aliases or names to tape devices and groups of hosts to make life a little easier. You can refer to the sample `backup.config` file provided with the distribution and to the man page `backup.config(5)` for more specific details. Note that once you change the file, the changes take effect right away. It is important to carefully think out the changes that you are about to make, and to check the file when you are done to ensure that you didn't accidently change more than you planned.

5.2.1 chains

```
chains 2
```

The `chains` entry is used to determine the number of backup chains that you want to support. The default is to support 1 backup chain, which is analogous to what BSD `dump` maintains in its usual usage. What is a backup chain? A backup chain consists of a set of related backups: a full save and the incrementals that record the files that have changed since that full save. The best way to visualize it is to consider BSD `dump`. It uses the `/etc/dumpdates` file to save information about when each level was done on each file system. When you do a level 1 backup, the dump program uses the dumpdates file to determine when the level 0 backup was done, and then it saves all of the files that changed since that time as the level 1 backup. So a sequence of backups: a level 0, 1 and 2, respectively contain all of the files, all of the files which changed since the level 0 back, and all of the files which changed since the level 1 backup. Note that the level 1 and 2 backups depend on the level 0 backup - if the tape that contains that backup is lost or damaged, you can't do a full restore of the file system using just the level 1 and 2 backups.

The idea of multiple Independent backup chains is that we can use several dumpdates files (one for each chain) to create several sets of full saves and their dependent incremental backups that do not depend on each other. This way if a tape is lost or damaged we can fall back on one of the other (hopefully intact) backup chains and restore the file system to a reasonably up to date state.

You might want to start off with 1 chain and move to 2 chains at some later point - it is easy to do. Adding chains should never cause problems. If you reduce the number of chains, the backups associated with the old chains will have the default keep values applied, which might make them expire sooner or later than they would have with their original keep values.

5.2.2 level

The `level` entries are used to define backup levels that you want to use in your system. Each level definition applies to a particular chain, and defines the level number (similar to `dump's` notion of a level), a name, and values for `every` and `keep`. The values of `keep` and `every` determine how long the backups stick around on tape before the tapes can be reused and how often the backups of that level should be done. For example, a daily backup would have an every of 1 (day), a weekly would have an every of 7 (days), and a full save that was to be done monthly might have an every of 28 (days). You would then set the keep values for each of these levels to indicate how long the backups should remain valid. You might use 14 (days) for the dailies (to keep the last 2 weeks worth of daily backups), 28 (days) for the weeklies (keep a month's worth), and 365 (days) for the full saves (save them for a year). In general, `keep` should be at least twice the value of `every` to make it less likely that you will be using the latest daily tape to do today's daily (if you did that, there'd be a small window of time while the backup was in progress where you'd have neither today's daily nor yesterday's).

Here's an example:

```
#     chain level name   every keep
level 0     0     full   28    365
level 0     2     weekly 7     28
level 0     9     daily  1     7
```

```
level 1     0     full    7     14
level 1     2     daily   1     2
```

This defines 5 levels in 2 chains. Note that you can reuse the names in each chain, and that they do not have to have the same values for `every` and `keep`. In this example chain 0 has full saves that are done every 28 days, weeklies that are done every 7 days, and dailies that are done every day. Chain 1 has full saves that are done every week, and dailies that are done every day. Note that the chain 1 backups are kept for a minimal amount of time - this reduces the total number of tapes needed to support 2 chains but still provides the security of having a second chain.

You should set the `keep` values to retain backups that you want to refer to for restores. We often find it convenient to be able to retrieve my .login file from up to a week ago, so we set the `keep` for our chain 0 dailies to 7.

5.2.3 location

```
#               name
location        bermuda
location        hi308
location        hi308d
```

The `location` entries declare tape drive and tape storage locations for use in the `offsite` and `tape` entries.

5.2.4 offsite

```
#       location        chain   wait    warn
offsite bermuda         1       2       1
```

The `offsite` entry is used to define an offsite tape storage location. Each backup chain that is involved in offsite tape storage is associated with a specific location, so that all of the tapes for that chain end up in one place - an important consideration in the event of a disaster. You can associate more than one chain with each location. You also define the number of days to wait before sending newly written tapes to the offsite storage site (you might want to retain them locally for a few days for simple restores), and the number of days advance warning to give before tapes will be freed (so that you can bring them back to the local site on the day that they will become free for general use again).

I suggest using at least 2 backup chains and storing the tapes from one chain offsite (for disaster recovery) and one chain locally (for general restores).

5.2.5 group

```
group a   foo bar baz
```

The group entry defines a group of hosts. In the example above we define a group named "a" that includes hosts "foo", "bar" and "baz". You don't have to define groups, but they can be convenient. A host can belong to as many groups as you want.

5.2.6 tape-type

The `tape-type` entry defines a type of tape. It looks like this:

```
#         type  age   uses formats
tape-type exb   2555  50   8200 8500
tape-type stone 99999 1    granite
```

This defines two tape types: "exb" and "stone". Type "exb" (for Exabyte) tapes can be up to 2555 days old and used 50 times (note that these are numbers pulled out of thin air!), and it supports 8200 and 8500 formats. Type "stone" lasts a looong time, can only be used once, and comes in one format named granite.

5.2.7 tape

The `tape` entries are used to declare names and definitions for tape drives. Some sample entries:

```
#    name user@host:dev          type reads      writes status location
tape 2    archive:/dev/nrmt8     hit  1600|6250 6250   yes    hi308d
tape 3    abyss:/dev/nrmt0       hit  1600|6250 1600   yes    hi308d
tape 4    abyss:/dev/nrmt8       hit  1600|6250 6250   yes    hi308d
tape 5    brachiosaur:/dev/nrst0 qic  11|24|150 150    no     bermuda
tape 6    favorite:/dev/nrst0    exb  8200      8200   yes    hi308
```

This defines several tape drives. Each drive has a name (which can be any string of non-whitespace characters) and a type, and includes info about which host and device the drive is. It also declares the formats that this device can read and write. If the drive will automatically read several formats, you can give the list separated by "`|`" characters. The "status" should be `yes` if that host has a `mt status` command that works properly (e.g. it is a Sun running SunOS 4.x or Solaris 2.x), and `no` otherwise.

5.2.8 config-name

You can give a name to this incarnation of the backup system using the `config-name` entry. The default configuration name is "", in which case everything acts like it always has been. You can set the config name to something like "test" to set up a test installation on a host that is running a production version of the backup system, so that your test version won't interfere with it.

The configuration name is used in various places to ensure that multiple incarnations of the backup system don't step on each other. For example, if the config name was "test", tape IDs would start with "test-" to avoid confusion with normal tape IDs. The `.chain` files would include "test" in their names to distinguish them from the others. The `dumpdates` file would include "test" in their names. Lock files created in `/tmp` would include "test" in their names.

This makes it possible to install a test installation of the system in conjunction with a production version to do more comprehensive testing before switching to it. It also makes it easy to set up several distinct systems on different sets of hosts and not confuse the tapes with each other.

To set up a test installation, go through the installation procedures but set things up in separate directories. The directory for the scripts should be different than your production version, and you should certainly make the database directory different. Make sure that you change the `Makefile`, `global.defs` and `local.defs` to reflect the correct locations of things for your test installation. Then edit the `backup.config` file and use the `config-name` entry to give your test installation a name, like "test". You might also want to change the value of `$backup_file` in `local.defs` so that you can backup a different set of file systems in the test configuration on each host.

Once you've tested a new copy of the system to your satisfaction, you should be able to simply copy the new scripts and binaries to your production directory, update your production `backup.config` file and continue working with your existing production database.

Once you have added tapes to the database, you should not change the configuration name. This is because the name gets incorporated into the tape IDs that are used in the tape and backup databases, and in the tape labels on each tape.

5.2.9 Miscellanea

You can set up a default tape drive with the `default-tape` entry. You can also define a list of addresses to mail reports to with the `mail-to` entry.

5.3 $db_dir/backups

The `$db_dir/backups` file determines which filesystems should be backed up and what program to use to back each of them up. You can also use this to control which backups to do on this file system, by chain and by level. For example, I could set up a file system to use `dump` to do all chain 0 backups and GNU `tar` to do only chain 1, level 0 backups with the following setup:

```
host:/export/0 dump chain=0
host:/export/1 gtar chain=1,0
```

You can also define where online backups should go, and you can assign one or more tags to each file system. See the `backups(5)` man page for more details.

I've added the ability to use `host:directory` rather than just `directory`, so that you can use a single `backups` file (usually located in your database directory). This will probably be required in future releases of the backup system.

You can also add entries for file systems that you do not want to backup by using `none` as the backup method. This allows you to add entries to keep `check-backups` from complaining about file systems that are in `/etc/fstab` but not in `$db_dir/backups`.

6 Doing Backups

Once you've installed the software and set up the configuration files, you should be ready to do backups. The first thing that you need to do is add tapes to the tape library. After that, backups consist simply of using the `all-backups` program to initiate backup runs.

6.1 Adding Tapes to the Tape Database

Tapes must be added to the tape library before they can be used in the backup system. The `add-tape` program is used to add tapes to the library. This program writes a tape label at the start of the tape, which is used to identify the tape automatically, and creates an entry for the tape in the tape database.

Each tape gets a tape identification "number" (tape ID). By default, the tape ID for a tape is its type followed by a number, such as "exb-25" or "tk50-2". If you used the `config-name` entry in the `backups.config` file to give a name to this backup installation, then that name will be prepended to the tape IDs for that installation (for example, "test-exb-25" if `config_name` is "test").

You can override this naming scheme by telling `add-tape` to use a chosen prefix string with the `-p prefix` option. If I were to run `add-tape -t 5 -p foo` the tape ID of this tape would be "foo-5". This might be handy if you have your own tape labeling and storage scheme.

I'd suggest marking the label of each tape in ink with the tape number, and using pencil to write other identifying info (hosts, chain, level, date, tape drive, run ID number). That way you can reuse the tapes easily by erasing the penciled in stuff, and you won't have to worry about relabeling the tape with the tape number every time.

I also suggest that you keep a paper log book that records who did which backup, what the Run ID number and tape numbers were, which hosts, what date, chain and level, and so on.

Normally `add-tape` chooses the tape number for the tape, but you can specify a tape number using the `-t` option. This can be useful if you want to replace a lost or damaged tape, or if the tape label on a tape was damaged somehow. You can also specify a range of tape numbers with `-t n:m`, which would cause `add-tape` to add tapes with numbers `n` through `m`.

`add-tape` is fairly safe to run - it will warn you if you try to add a tape with an ID that already exists in the database, or if you try to add a tape that already appears to have a tape label.

6.2 Starting a Backup Run

`all-backups` is the program you want. If you start it with no command line arguments, it'll ask you for the information it needs. As you get more used to it, you can use command line arguments to get things done more quickly.

I'd suggest starting with a full save on one machine. Heck, do a few full saves on different machines. Then try an incremental on those machines. If you tell it to do the backup "now", the backups will be done in the foreground (so to speak), where you can watch them. If you specify a time, `all-backups` will reschedule the backups to be run at a later time with the `at` command, and you won't see the output until the backup is over and the results are mailed to you.

As you get used to how things work, you can start exploring the power of the system. Try using "group aliases" to do incrementals on the hosts that you did the full saves on. Use the rescheduling abilities to run backups in the background automatically.

When you feel really confident, try out the backup during boot option. Careful - your system will go down at the time that you scheduled the backup for!

If you are really daring, try using the `-p` option to do an incremental on several hosts in parallel mode. This will cause the backups on those hosts to run concurrently, writing their backup files to disk rather than to tape. As they finish, `all-backups` will copy the backup files to tape and update the database. This allows one to do backups on lots of hosts somewhat more quickly than you would normally be able to do. This is especially useful if you are also using the `-m disk` or `-m both` modes, since they have to do the backups to disk anyway.

In a large installation where you cannot fit all of your daily backups on one tape, you'll probably want to do something like what we do: create several groups of hosts, each group will be done together for incrementals in the same backup run. We have 6 tape drives, so we created 6 groups with about 6 hosts in each, and when we do our daily or weekly backups, we just start 6 copies of `all-backups`, one for each group. Works great. We made it simple to set up by writing a shell scripts named `weekly` and `daily` that start up 6 windows running each of the all-backups, with all the arguments set. Each window pops up and tells you what tape to mount in which drive. That way you can get the tapes, label them, pop them in the drives, and hit return in each window and poof, your backups have been set up. You can find a copy of our scripts in the `cis` subdirectory.

The current version is limited to 1 backup run per tape - if you hit the end of tape the run stops. This is because none of the currently supported archiving programs really deal with end of tape correctly, in that there's no way to coordinate tape use with the backup scripts and the archive programs without doing really absurd things. Eventually I hope to change this, but in the meantime, the limitation exists and you have to live around or with it. Unfortunately, we recently set up 3 servers that have about 4 gigabytes of disk space each, which certainly won't fit on out Exabyte 8200 tapes, so I've added some stuff to help you work around the one tape limit.

If you need to backup a host whose file systems won't fit on a single tape, you can use the `tag` facility in the `$db_dir/backups` file, and the `-i tag` and `-x tag` options in `all-backups` as a cheap workaround. Suppose I want to break up the backups on a host so that file systems A, B and C are on one backup run, and D, E and F are on another. Create a `$db_dir/backups` file that looks something like this:

```
host:/A  dump tag=first
host:/B  dump tag=first
host:/C  dump tag=first
host:/D  dump tag=second
host:/E  dump tag=second
host:/F  dump tag=second
```

If you start a backup run with `all-backups -i first`, then the run will cover every file system on every listed host that has a `tag=first` entry.

We often ran into a problem where we'd schedule a backup for 6 A.M., and someone would remove the tapes from the drive before then and forget to replace them. This made the backups abort, and that was a drag. We've added a feature that allows you to schedule a tape check for some time while there are still people around to fix the problem. Check out the `-C` option to `all-backups`.

If you are concerned about losing a backup chain if someone trashes a full save tape or something, then use multiple chains. If you're worried about relying on `dump` or GNU `tar` for all of your backups, use `dump` to do your backups in one backup chain and GNU `tar` to do backups in the other.

6.3 Checking a Backup Run

Using the rescheduling option can be somewhat confusing. The `atq` command is useful for viewing the `at` queue - run it on the host that you ran `all-backups` from to see whether the `at` job got scheduled and what time it was scheduled for. A common mistake is to schedule a job for 5:00 A.M. when you meant to schedule it for 17:00. Note also that some versions of `at` only run their queues every 15 minutes, so if you schedule something for 10:10, it might actually run at 10:15. Note also that some versions of `at` will only run 2 or 3 jobs at a time. If you have such as system and schedule 6 jobs for the same time, several of them will wait until the running jobs finish. There are sometimes ways to change the limits - consult your man pages and reference manuals.

You can also use the `show run` command to list the database entries corresponding to a given run identification number. This will list all of the pertinent entries from the online databases and might give you some insight about what's finished and what's not. The `show scheduled` command will give you a list of the backup runs that have been scheduled or started in the last 24 hours but which have not finished yet.

See the section on Problems (section 12) for more suggestions about problem solving.

7 Doing Restores

The easiest way to restore files from the backup tapes is to use the `getback` program. `Getback` provides you with 3 modes of operation: full restore, interactive restore, and pick restore. The full and interactive restore modes are similar: you specify the host, directory and date that you want to restore as of, and `get-back` will walk you through restoring the files from what it thinks is the best sequence of backups (starting with a full save, working toward the most recent incremental). Full restore mode extracts the whole contents of each backup into the current directory, whereas interactive restore mode allows you to select which files to extract ala the BSD `restore i` option (and so it only works for `dump` archives, and not for GNU `tar` archives). Pick restore mode gives you a list of all of the relevant backups from the database and allows you to select a backup and extract files in interactive fashion from each backup. Again, this only works for `dump` archives.

`getback` will check each tape that is loaded to ensure that it is the correct tape, and will skip to the correct file on the tape before running the appropriate restore software to extract the contents. So all you need to do is tell it the host, directory and time, and load tapes when it asks. If the backups are also located online, `getback` won't bother with the tapes but will instead just retrieve the data from the online files.

You can also restore files by hand, of course. Use `show` to list the entries for the backup run that you want to restore from. Find the tape number and file number of the backup you are interested, and mount that tape in a tape drive. Use `mt fsf` to skip to the appropriate file (tape files are numbered starting at 0, which is the tape label). Then run the restore command of your choice to extract the contents of that backup.

For example, we might see something like this from `show run`:

```
  33   0,0      done  yes     660944047     660950541 solar
 solar      buid 14   done       tape   6.1 /
 solar      buid 15   done       tape   6.2 /export/0
```

This is an entry for backup run number 33, which was a chain 0, level 0 backup on host "solar". Two file systems were backed up: "/" which is on tape 6, file 1, and "/export/0" which is on tape 6 file 2. Mount the tape, skip to the file, and go.

You can do similar things with the output from `list-backups`.

8 Tape Management

As previously mentioned, the `add-tape` program is used to add tapes in the tape library and label them with a "magnetic" tape label. You may find it necessary to relabel tapes whose magnetic labels have been destroyed - you can use `add-tape -t #` to do that.

The `run-clean-db` program should be set up to run nightly. It will automatically take care of the various on-line database, and it is what controls the recycling of tapes as the backups that they contain expire. If you want to force a tape to be reused more quickly than `run-clean-db` would free it for use, you can use the `reuse` command. Be careful, you could easily "reuse" the wrong tape and lose all of the backups that are stored on it.

After a long while, the tapes themselves will expire, either because they are older than the age set for that type type of tape or because they have been used more than the maximum number of times. At that point you should replace them with fresh tapes.

If you are using offsite tape storage, you will be using the `offsite` program to generate a report that describes what tapes to take to each offsite location and which tapes to bring back. The report will describe, for each offsite storage location, which tapes should be there, which tapes should be taken there, and which should be brought back from there.

The tape database keeps track of the location of the tapes. Each tape has a home location, which defaults to the location of the tape drive used when the tape was added to the tape library with `add-tape`. You can change the current location of a tape with the `move-tape` program.

9 The Graph Making Stuff

This backup system includes some scripts that can be used to automatically generate graphs using `gnuplot`. These graphs include general overviews of run size by rank and run ID, run duration by rank and ID, tape uses, and graphs of backup sizes by time, per file system. This is an optional portion of the system, so you don't need to install it. You might find it interesting, however.

You can find a brief description and installation instructions in the `graph` subdirectory.

Here are 3 examples of the sorts of graphs that you can produce.

Figure 1: Backup sizes ranked by time for fish:/export/0.

Figure 2: Run sizes ranked by size and divided by level.

Figure 1 shows the sizes of the backups for one file system plotted by time. The different levels are distinguished with different line styles. This is from the `Merged` directory, so it shows all of the backups that have been done since I created the graph system (and got it working right).

Figure 2 plots the total size of a backup run (the sum of the sizes of the backups in the run) sorted in rank order from largest to smallest. The data is segregated by level, and each level is plotted with its own line style on top of each other for comparison. This graph also represents a summary of all of the data that we have collected. This shows that most of our runs are less than 500 megabytes in size, well within the limits of the 2.3 gigabyte tapes that we are using.

Finally, figure 3 shows the duration of each run plotted against its rank. In this example, the levels are plotted with their own line styles, but ranked to-

Figure 3: Run duration ranked by time.

gether. This graph is from the `Recent` directory, and it shows the data collected from runs in the last month.

10 Daily Procedures

Just to keep things in perspective, here's a list of the various procedures that should be performed on a regular basis. Most of these will probably be set up to run via `cron` or some such method, though the backups will probably be initiated by hand. Most of these will probably be done daily.

- Sometime in the wee hours of the morning `run-clean-db` will run and delete old backups from the database and free the associated tapes for reuse.
- Collect the tapes from the previous night's backups, write protect them, rack them, and update the backup log book.
- If you are bringing tapes back from off-site, use `move-tape` to check them back into local storage. Use `move-tape` to move the tapes that you took off site yesterday to their off site locations also.
- Print the `list-backups` report, and add the printout to the backup log book. If you are using off site tape storage, make a copy and send it off site for safe keeping.
- Read the `check-backups` warnings from all of the hosts (if any) and schedule backups to correct the problems found.
- Review the `backup-report` chart, investigate any gaps. Use this to check whether you are getting the `check-backups` warnings that you should.
- Check the report given by `offsite`, gather the tapes to be taken offsite.
- Schedule and set up backups that are to be run later in the day.

11 Disaster Recovery

There are disasters, and then there are disasters.

If a host has died and needs to be restored, then there are two ways that you can go about doing this. If you can get the host to boot to the point where the backup system software is available, then you should just be able to use the `getback` program to restore each file system. In our case, its usually possible to boot a host as a diskless client of another machine, from which point we can use the `getback` program to restore everything. If you cannot access the backup system software from the broken machine, then you can use `getback` or `list-backups` on another machine to get the list of backups that you will need to restore from, and then run the restores "by hand" by mounting the tapes, skipping to the right files and recovering the needed backups.

If the database host is dead, or the backup database is otherwise unavailable, then you can use the printed copies of the `list-backups` output (you DID make printed copies, didn't you?) to find the crucial backups for each of the file systems that you want to recover. You should also be able to use your backup log book, though that might entail a lot of extra work since the backups won't be sorted by file system. Failing that, you could always read the physical labels that you stuck on each tape.

The worst disaster would be if your entire site were destroyed, including the backup tapes, log book, list-backups output and everything. If you were using offsite tape storage, then you can use them to recover your files. I would duplicate the tapes, keeping a copy in safe keeping at the offsite tape storage location. Take the copies to your new site and use your off site copy of the list-backups output to determine which backups to use to restore which file systems. You probably want to start with the backup database, since once that's restored you can use it to do the rest.

In summary, to help ensure complete disaster recovery, you should:

- Use offsite tape storage.
- Maintain a copy of the list-backups printouts at each offsite storage location.
- Maintain a backup journal that describes each backup run, the tape used, the chain and level numbers, the hosts, the date and the status.
- Label each tape to indicate its current contents, including chain and level number, hosts, and date.

12 Problems that Crop Up

There are (of course) zillions of things that could go wrong when you try to do your backups. This is a description of the most common problems and some suggested solutions.

Sometimes things seem to lock up - you schedule a backup, but it doesn't. Or it starts, but doesn't finish when you expect (or at all). You may have a problem with locking. Sometimes stale locks are left in the backup database directory, named .lock. If there are no other backups running now, and there are lock files present, you could just delete the existing lock files and see whether that frees things up. The lock files contain information about the process and host that created them - you can use that to find out whether the lock owner is still running. I've noticed at times that the server that hosts the backup database gets confused and won't allow other hosts to make the locks, even though the lock files don't exist. Sometimes rebooting will help that. There's also a funky problem with `rsh`. If you `rsh` a command to a remote host and that host crashes, the `rsh` will hang for a long time (forever?) and `all-backups` will hang waiting for the `rsh` to exit. Use `kill` to nuke the offending rsh's and `all-backups` should continue. You might need to clean out stale locks that were left behind by the crashed host. Lock files for the `dumpdates` files are in `/tmp` on each host.

If you use the "reschedule with at" option to start a backup in the background at some future time, make sure that the future time is far enough in the future that `all-backups` has time to check the tape label and stuff before that time arrives. Otherwise `at` will either reschedule your job for the next day or tell you that its too late to schedule the job. Use `atq` to check the `at` queue. Check the time AND the date that the jobs are scheduled for - its easy to get confused and schedule a backup for tomorrow at 5 A.M. when you really meant for it to run today at 5 P.M. (use 24 hour time).

If you schedule a job and it disappears from the queue but didn't send mail about the results, there are several things you can look for to find more information. The output is collected in a file named `/tmp/backup-output.NNN` where `NNN` is the run ID for that job. You can read that to see what had been done so far. If that isn't there, its possible that the job didn't even start correctly, or that it ran but that the mail system is screwed up. Check wherever it is that mail for mailer-daemon and root go to - see if there's any informative notes there. Make sure that your mail system is working right.

If you set `$use_deadman_stuff` to 1, then you can use the "deadman switches" to kill backups that are taking too long. If this variable is set, `all-backups`, `do-backups`, `backup-dump` and `backup-gtar` start up background processes that periodically check for one of several files in `$db_dir/deadman`. If one of the files appears, then the process sends a hangup signal to the program that started it and exits. All of these programs look for a file named after the run ID for that run. In addition, `do-backups` will die if a file named `hostname.rXXXX` is created (where XXXX is the run ID) and `backup-dump` exits if a file named `hostname.bXXXX` is created (where XXXX is the backup ID number. The deadman processes for `backup-dump` and `backup-gtar` can also be used to kill the backup if a certain amount of time has elapsed. You can set a timeout for backups on a per filesystem basis by using the `timeout=NNN` option in the `backups` file (NNN is the number of seconds allowed for backups on that directory). See `backups(5)` for details.

Don't rely on `check-backups` to schedule your backups - it is meant to be a safety measure, not a primary backup scheduler. Make a schedule, write it on a calendar, and try to stick to it.

You might want to check periodically that `check-backups` is running correctly on each host. Look over the `$db_dir/backups` files while you are at it.

Remember that you can't use the "backup during boot" option when you have specified several hosts, and that you have to schedule the backup on the host that you are rebooting.

Be careful with reuse - don't reuse the wrong tape!

13 Archive Programs

One of the advantages of this backup system is that it doesn't restrict your choice of archive programs too much. The archive programs should support full and incremental backups in some form, and there should be a way to pass information about the tape device or archive file to use, and the directory to backup on the command line. The system comes with support for BSD dump and GNU tar built in. You should be able to easily add support for other programs, like normal tar, cpio or database backup programs.

Both dump and GNU tar are good general purpose archive programs. They each have their shortcomings, however.

Neither of them will work completely correctly on active file systems. dump is subject to more problems that GNU tar, since it is reading the file system "directly" rather than going through the OS. There are patches available from Purdue that make dump somewhat more safe to use on active file systems. Both dump and GNU tar will skip files that are being modified when they come to them, meaning that those files will not appear on the backup. It is also possible for files to move from places that the programs have not scanned yet to places that have already been scanned, in which case they will also not make it into the backup. For a more detailed analysis of the problems of backing up active file systems, see Steve Shumway's excellent paper "Issues in On-line Backup" (Proceedings of the Fifth Large Installation Systems Administration Conference, 1991).

There are also other problems with various archiving programs - see Elizabeth Zwicky's paper "Torture-testing Backup and Archive Programs: Things You Ought to Know But Probably Would Rather Not" (also found in the LISA V proceedings).

Sun has a product named Backup Copilot that solves this problem using a special set of locks that programs can invoke to freeze certain types of file system activity while they are scanning for changed files or reading files for a backup. You could either get that and use it for backups of your Suns (its a very nice backup system) or get it and use its version of dump with this backup system - it should be easy to incorporate.

If you don't have a means of safely backing up active file systems, you can either risk having bad backups, or do your backups in single user mode, or compromise. We compromise by doing all of our full saves in single user to ensure that we have at least one good backup a month, and then do many, many incrementals (2 per day for each file system) with the systems live.

GNU tar is nice in that the archives are more portable, and you can backup by directory rather than by file system. It doesn't support all of the strange file system objects that you can find, such as HP context dependent files or ACLs. It does support backup and restore of sparsely allocated files (such as core files), but does it by reading the file in 512 byte blocks (which is painfully slow for large sparse files), and may not get the sparse blocks in exactly the right places. If you decide to use GNU tar, I strongly suggest that you get at least version 1.11, since previous versions didn't correctly support deleting restores (a full restore where deleted files are deleted, if that makes sense. Its hard to explain).

GNU `tar` does not have a nice interactive restore program like the BSD `restore -i`. You are restricted to extracting the whole archive or the files that you name on the command line. In that respect `dump` is extremely handy - you can run `getback` and use it to mount and check the tapes, and skip to the right files. At that point it will run `restore -i` and you can poke around to see if this backup contains the files that you want to extract. If not, quit and `getback` will go on to the next backup.

13.1 Adding Support for New Archive Programs

It should be fairly easy to add support for new archive programs. Each archive program `foo` should have a `backup-foo` and a `restore-foo` program that the backup system uses to do backups and restores. In some cases (e.g. `dump`) these programs are little more than wrappers to call the appropriate backup or restore program with suitable arguments. In other cases (GNU `tar`) these programs have to "map" the full save/incremental structure onto a program that doesn't directly support the same ideas.

The `backup-*` program is given the name of a directory to back up, the chain and level number of this backup, the name of the tape device or file to write the archive to, the backup ID number, and (if applicable) a flag that tells it that the archive is being written to disk rather than tape. It needs to convert that information into whatever it needs to run the given archiving program. In the case of dump, that means that it uses the directory to find the device name, it uses the chain to select one of several `dumpdates` files to use, and it passes the level straight to dump. In the case of GNU `tar`, it uses the information available in the backup database. In both cases it updates the database entry for this backup with the dates of the previous backup and the date of this backup, sets up the output file, makes a command line, and runs the archiving program.

I am planning on changing `backup-dump` to work in a fashion similar to `backup-gtar`, in that it will gather information from the backup database and create a fake `dumpdates` entry to control the actions of `dump`, which will eliminate the possibility of the backup database and the dumpdates file getting out of synch with one another.

Similar approaches could be made to incorporate other archive programs. For example, a `backup-cpio` program could use the dates to run find, printing a list of files modified since that date and pipe that into cpio with appropriate arguments set. I would use `backup-dump` as a template and work from there.

The restore-* programs are simpler. They are passed command line arguments that tell whether to do an "interactive" style restore (ala BSD restore -i) or a full restore, the name of the tape device or archive file to read, and optionally a list of files to extract from the archive. The only complication is that not all programs support an interactive restore mode. In the case of tar, we just do a normal full restore for both modes, and assume that the user will give a list of files on the command line to get the specific files that they want.

Once you've made your backup and restore programs, you will need to add the name of that archive type to the list %valid_archive_types in global.defs. Then you can set up entries in your $db_dir/backups files to use the new archive method.

14 Future Plans

I have zillions of ideas for things to add. Multiple tape support. Tape rollover. Copy the database to the end of each backup run for database disaster recovery. Provide support for "strange" media like WORMs or jukeboxes. Keep information about the contents (files and directories) of each backup on-line so that you can easily find the most recent backup that contains file X, and so that you can do user-initiated restores.

14.1 Multiple Tape Support

Multiple tape support is difficult. There are several potential problems that one can run into.

The first is that not all archive programs support multiple tapes. This isn't a huge problem though, since the backup system could still support multiple tapes and leave it to the user to ensure that they don't use a bogus archiving program with multiple tapes.

The second problem is that programs like dump don't do end of tape (EOT) detection, but do EOT prediction. This means that dump needs to be told how much data it can write to the current tape. This worked to some degree in an environment where you were writing one backup onto several tapes, but fails in the case where you are writing multiple backups to multiple tapes. It is difficult for the backup scripts to keep track of home much space is left on the tape, since there's no standard way for the archive programs to report that information or to get it from the tape device. And even if you did tell dump how much space was left on this tape, it would assume that that applied to successive tapes as well, so if dump started on a tape that had only 10 megabytes of space left, it would assume that successive tapes could only hold 10 megabytes of data.

The third problem is that even if a program supports real EOT detection and so avoids the space prediction problem, it may not accept more than one tape device on the command line, meaning that it will have to interact with something through /dev/tty or stdin to get the name of the next drive to use, or to request that the next tape be mounted. One could "solve" this problem by interpreting the output from the program and supplying appropriate responses, but this complicates the handling of stdout and stderr from the archive program. Not all programs will allow you to switch to a different drive, let alone to a different drive on a different host, and so you might be stuck with human interaction anyway to mount the next tape, unless you have a tape stacker or jukebox.

In other words, it is a mess.

There are several approaches that I could take in addressing this problem:

1. Ignore programs that can't handle proper EOT detection, and either make use of the special features in `GNU tar` that allow you to call a program when it needs a new tape, or have backup-gtar interpret the output from the program and do the right things in response.

   There are versions of dump that support proper EOT detection (the latest BSD version, for instance). I could leave it to people to install (and test) that if they want, and solve the problem in a case-by-case fashion (interpret the output of `dump`, use the program option with `GNU tar`).

2. Fake the programs into thinking that they have an infinite tape to write to. This is the approach used in DeeJay, a backup system developed at Columbia University (see "DeeJay - The Dump Jockey: A Heterogeneous Network Backup System", by Melissa Metz and Howie Kaye, Proceedings of the Sixth System Administration Conference, 1992 (LISA VI)). Each archive program writes its output to a proxy somewhere, which handles EOT detection, multiple tape handling, database updates and so on. It seems like a nice solution that would work transparently for almost any archive program. It has the added advantage that you should be able to improve the performance of writing the backup over the net since you should be able to avoid the synchronous nature of the normal rmt protocol. You can also have the proxy handle switching between tape drives, even on different hosts, so that you can use tape rollover instead of requiring a stacker or jukebox.

I suspect that I will pursue the second option.

There's also the question of providing support for tape and disk stackers and jukeboxes. I don't have any of the hardware to play with, so my understanding of these devices is all based on hearsay and I don't really have a broad understanding of how people use these devices. We'll need to be able to identify tape drives as stackers or jukeboxes. For backups, `all-backups` will have to change to choose free tapes that are in the queues of the drives named for that run. That's easy, we can create special tape locations for each queue. The vaporware that deals with multiple tape support will have to know how to change tapes on these devices, and I don't imagine that there's a standard command interface for these devices (that would be too simple...) so there'll have to be a level of information hiding in the code somewhere. Dealing with cases where the tapes needed for a backup or restore aren't in the queue for a drive will be more complicated. I imagine we'll need a program that will make suggestions about what to load in the queues and what to remove (akin to the playlist program mentioned in the DeeJay paper). I suspect that there are some differences in how people use different kinds of autochangers - whether to keep it mostly full of free tapes so that you can do a weeks worth of backups unattended, or whether to keep it mostly full of recent backups so that you can do restores more readily. I have more questions than answers at this point.

14.2 Control of Incrementals

The idea behind incremental backups is simple: save everything that has changed since some time X in the past, which is is when the more recent lower level backup has been done. This is obviously useful, since it usually greatly reduces the amount of data that you have to save to make a complete copy of the current file system contents in backups. BSD `dump` implements this through the `/etc/dumpdates` file. This contains an entry for the most recent backup of each disk partition at each incremental level, giving the date of the backup. When I tell `dump` that I want to do a level 1 backup, it consults the `dumpdates` file to find the date of the most recent level 0 backup for that file system, and saves everything that has changed since that date.

There are two problems with this approach. The first is that to implement multiple independent backup chains, you need to use multiple copies of the `dumpdates` file. This is somewhat messy, and adds an extra bit of complexity since the backup system needs to juggle the various versions around.

The second is more insidious. The information in the `dumpdates` file is a duplicate of stuff that can already be found in the backup database. Its possible for the two to get out of sync, since `dumpdates` is maintained by `dump` and the backup database by the backup system. For example, if I were to do a level 2 backup, the `dumpdates` file would contain the date of that level 2 backup (call that "Thursday"). If I were to accidently reuse or otherwise destroy that tape, the backup database would correctly adjust itself by removing that backup (since it no longer exists and cannot be restored from). Subsequent level 9 dumps will still be done as though the Thursday level 2 backup had been done, and won't save anything that changed before Thursday.

I have already change `backup-gtar` to fix this problem. I will be doing something similar with `backup-dump` in the future.

It would be nice if archive programs could be told a date to act on, like GNU `tar`, since that makes it much easier to control them.

14.3 Online Catalogue of Files

The backup system makes restores easier by helping you find the backups that have been done, and to select the most appropriate backups. It also deals with requesting the tapes, checking to see that they are correct, and skipping to the right files on them and running the extraction programs for you.

It would be nicer still if you could identify the backups that a specific file appears on so that you could avoid mounting and searching unnecessary backups. This could be done by storing a catalogue of files saved in each backup online somewhere, and writing something that would allow you to browse that and select versions of files to restore. Shoot, you could get fancy and provide for user initiated restores. The problem is that you need a standard way of representing the online catalogue that would work and be supported by all of the archive programs that you are using.

14.4 Backup of Active File Systems

It is impossible for an archive program to make a complete and correct backup of an active file system without some sort of support from the operating system. This is why people recommend that you do backups while the file systems are quiescent, like when the system is in single user mode.

I consider this to be an important problem, but one that is outside the scope of the backup system, since I can't and won't be providing any operating system patches to support safe backups of active file systems. Sorry. If you are using Suns, consider getting a copy of Backup Copilot and using their dump program, which uses special locks to ensure that it gets a correct snapshot of the file system. If you are using System V, you can talk to Veritas about getting a copy of their file system that provides copy on write features that you can use to make safe backups of active file systems. AFS from Transarc also provides a copy on write facility, with their own incremental backup facility.

14.5 Interoperability

I think everyone would agree that it would be nice if archive programs could create archives that can be extracted onto other (possibly very different) systems, and have reasonable things happen. All too often it is difficult or impossible to extract an archive written on one system onto another system (for example, try reading a Pyramid dump onto a Sun). Programs that use a "standard" format that can be read on many systems don't always support system dependent file types. For example, `tar` doesn't support sparsely allocated file or HP context dependent files or access control lists.

An ideal archive program would allow one to store any type of file in some standard format which could be read on all platforms. The extraction process could either ignore parts that don't correspond exactly to objects supported in the local file system (extracting context dependent files on Suns, for instance), or to do the best job possible (extracting a sparsely allocated file onto a system that doesn't support them would just create them with the holes filled in).

Oh, that would be nice. You could consider doing this with a dump-like program that writes a `dump header` for each file into the archive, as well as a generic "pax" style header. The restore program would have to be smart enough to deal with a wide variety of file system objects, and one would have to find a way to distribute and incorporate new extraction routines when they become available. I suppose the default handler could just ignore unknown file types or something.

A The Backup Database

The backup database seems complicated due especially to the number of files involved, but it isn't really that bad. Conceptually, think of it as 2 separate pieces: an indexed list of tapes (the tape library) and a hierarchical graph of backup runs and backups.

A.1 Tape Database

The tape database has an entry for each of the tapes in the library. Each entry is tape ID number, which looks like "exb-123" (tape type and number) or "test-exb-123" (config name, type and number). Each entry has a number of fields.

flags The status of this tape. Valid values are "free" (free for use in a backup run), "used" (already used in a backup run), "promised" (promised for use in a later backup run) and "expired" (tape is tool old or has been used too many times). Tapes that are promised for backup runs are freed again after 24 hours - if you need to schedule backups more than 24 hours in advance, you might want to futz the timeout in `run-clean-db` so that backup runs later in the week won't try to use the same tape.

type The type of each tape is set when the tape is added to the database, and is always set to the type of the drive that is used to write the initial tape label. This field should never change.

format The format is more specific than the type. Type refers to the general type of tape - what sort of drive it will physically fit into (e.g. exabyte, half-inch, quarter-inch, tk50). The format might be the tape format (qic-11, qic-24) or density (800 bpi, 1600 bpi) or whatever. This is set according to the format of the last drive that was used to write this tape.

add_date The date that the tape was added to the database. Should never change. This is used to determine when a tape should expire by age.

run_id The ID of the backup run that this tape contains (if any).

drive The host and device name of the drive that was last used to write this tape. Useful to find the drive if you need to read the tape and the drives heads were out of alignment or something.

location The location of the tape. If this field doesn't exist, or if its value is "", the tape is local. Otherwise the value is the name of the offsite storage location where this tape is stored at.

used_date This is the date that the tape was last used, whether it was promised for a backup or actually used in one.

uses A count of the number of backup runs that this tape has been used for. Used to expire tapes by usage count.

comments Not currently used for anything.

location The current location of this tape.

home The home location of this tape.

Each tape also contains a tape label written as the first file on the tape. The tape labels look something like this:

```
xyZZy T# exb-123
chain 0 level 2 drive allosaur:/dev/nrst0 run 2189
pun,saqqara,archive,shape,disney,laverne,mad,iron
```

The first line is the most important - the magic "number" is used to see whether this is a tape label, and the tape number should correspond to the tape's ID on its physical label. The other lines are informational, written so that you can identify a tape's use if the physical label is erased or falls off.

A.2 The Run Database

Each backup run that you start with `all-backups` creates a unique entry in the run database. These are indexed by run ID numbers, which should be monotonically increasing, starting at 1. The complete information for a backup run is actually stored in several files: the run database file (typically named "run.db" in the database directory - see `global.defs`), the per-host run database files (typically named things like "run.db.mammal.db"), and the per-host backup database files (named things like "backup.db.mammal.db"). These are separated into different files to make the hosts somewhat more independent of each other and especially to reduce lock contention (since locking is done per file, rather than per record).

The run database contains an entry for each backup run that is started with `all-backups` (well, an entry for the backup runs that haven't expired yet, to be more accurate). These are indexed by run ID number. The per-host run database files are also indexed by run ID number, and simply contain a list of the backup ID numbers that were associated with each run. The per-host backup database files contain information about each backup in a run - what file system, what date, where the backup is located on disk and/or on tape, what the status was, etc.

Let's examine these more closely. The run database contains entries for each run, which include the following fields:

chain The chain number of this run.

level The level number of this run.

hosts The names of the hosts that are included in this backup run (comma separated list, no whitespace).

flags The status of this run. Valid entries are "started", meaning that the run has started, and "done", meaning that the run is done. Entries are not created until the run actually starts, so if you schedule a run to start in the future its run database entry will not appear until the run actually starts. There's no specific value for "failed" - you have to consult the individual backups on the run to see how they fared. If the run actually completed in the sense that it made it to the end of the program, the status is set to "done".

active This isn't used for anything currently. It is set to "yes" if the backups were done in multi-user mode, and "no" if they were done in single user mode.

start_date When the backup started (timestamp).

done_date When the backup finished (timestamp).

scheduled_for This is the string passed to the `at` program for scheduling. No attempt is made to interpret it and convert it into a real date.

Each host named in the "hosts" field may (but might not) have a corresponding entry in its per-host run database file. Its entry will be missing if the backup run could execute `do-backups` on that host for some reason. The per-host run database entries are simply lists of the backup IDs associated with the backups of that host in this run. There's one field for each entry: buid_list, which is a comma separated list of the backup IDs from that hosts per-host backup database that are part of this run. This might be empty if the backups couldn't be done.

The entries in the per-host backup database are a bit meatier. They are indexed by backup ID, and contain information about a single backup (of a directory or file system). The fields are:

run The run ID that this backup is a part of.

fs The name of the file system or directory that this is a backup of (should be the mount point if this is a file system, rather than the raw device file or whatever).

fromdate When this backup's data spans. If this is a full save, this is set to 0. Otherwise it is set to the time of the most recent next lower level backup, ala dump's levels.

todate The time that this backup started.

flags The status of the tape copy of this backup, if it exists. Valid values are "started", "done", "failed" and "aborted". "failed" means that the archive program exited with non-zero status, "aborted" means that the program noted that the tape had been rewound and so it gives up for fear of overwriting a tape.

d-flags The status of the disk copy of this backup, if it exits. Valid values are "started", "done" and "failed".

tflist The list of tape/file numbers that this backup can be found on. The list should look like this: "exb-123.3,exb-145.1" which means that this backup is on tape exb-123, file 3 (counting from 0) and tape exb-145 file 1. Since this backup system doesn't support multiple tapes at all, your tflist's should be one element long.

comments Not used for anything yet.

afile Where this backup can be found on line, if anywhere.

volhdr If you used GNU tar for this backup, this is a copy of the volume header.

type The type of archive program used to create this backup (e.g., "dump" or "gtar").

In addition to the information in the database proper, the backup system also uses the "files", "errors" and "tar-db-dir" directories to store additional information. The `tar-db-dir` directory is used to store GNU tar's incremental listings if you are using a version that supports `+listed_incremental`. The `files` directory isn't used for anything - eventually I intend to store indices of what files are in which backups to make it even easier to do restores. The `errors` directory contains a copy of the stdout and stderr from the archive program for each active backup. These are stored with names like "fish.23", which would be the output from backup ID 23 on host fish. These are used by "show -v run" to list the verbose run information.

Figure 4: Relationship between entries in the various database files.