Config: A Mechanism for Installing and Tracking System Configurations

John P. Rouillard and Richard B. Martin – University of Massachusetts at Boston

ABSTRACT

One problem that faces system administrators is how to install and maintain local configuration information on a large number of machines. Some previous approaches such as cloning [2] help, but they only provide a baseline, not ongoing configuration control. Other mechanisms such as Typecast [7], Hobgoblin [5], Scrape [3] or Mkserv [6] assist in the configuration process, and provide some support for ongoing maintenance. However supporting multiple system configurations is still troublesome. Also it can be very difficult to delegate system administration tasks. Insufficient logging of file changes can create a nightmare when attempting to find the cause of a problem.

The method we present uses rdist and integrates it with *make*(1) and the CVS version control system to provide the ability to delegate and log changes. End node users can make changes to their workstations, however all changes are logged, so that it is possible to see what has changed on a given machine when problems occur. When making changes that affect a large number of machines (e.g., amd automount maps, rc files) previous versions of the file are available in the CVS tree and can be retrieved and distributed in case of unforeseen problems.

Introduction

The University of Massachusetts at Boston has used rdist from a single configuration area to ease multiple machine maintenance. While under contract to to Siemens Nixdorf Research and Development in Burlington, MA[1], John had a chance to further refine the rdist system to add version control elements to the system. As an added feature of this merger, delegation of responsibility for changes to a file is feasible.

The main features provided by the config method are:

- Ability to track changes to any and all configuration files and binaries on a system.
- Files that are to be distributed can be required to pass sanity checks to reduce the chance of errors.
- Concerned parties can be notified of file changes via electronic mail, news, or real time messaging such as alphanumeric pager, inform, or zephyr.
- Responsibility for changing files can be delegated to end users with fine grained access control.
- Multiple authorized people can change configuration files without conflict.
- Retrieval of previous (read working) versions of files allows changes to backed out easily.

[1]Primary extension development was done there, some other features were added at other sites.

The main programs that are used are the CVS source code control system, rdist version 6.1, GNU make and a home grown perl script that generates rdist macro (class) definitions from a database file.

The config system embodies all local configuration information for a host. Any system file on the workstation that is different from the file in a stock installation is included in the config system. This includes modifications to rc files, the fstab and vfstab files, the password file, as well as binaries that are required for operation (e.g., the amd automounter).

The implementation described in this paper is based on rdist, and therefore uses a push paradigm, but other distribution mechanisms such as *track*, *sup*, or *rcp* could be used.

The Components

Figure 1 displays the primary components of the system and the relations between them. The first component of the system is the ''Rdist Master CVS Tree'' hereafter referred to as the rdist master tree. This tree contains all local configuration information for a host.

In effect, the rdist master tree contains all of the information needed to tailor the vendor's distributed filesystem hierarchy to local requirements. This information embodies all of the ''personality'' of the workstation. Distribution to the workstation is controlled by a master distfile that resides in a subdirectory of the rdist master tree.

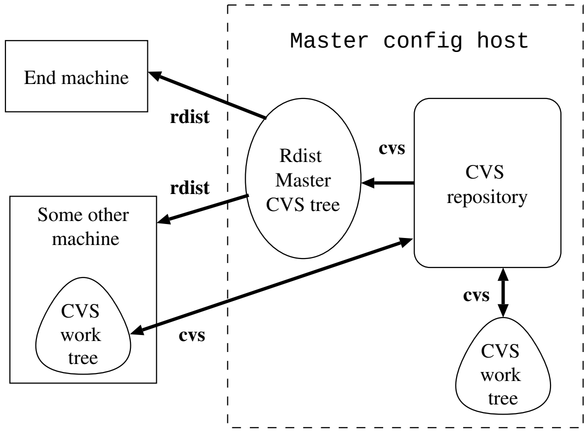

Figure 1: The components of the config system. The heavy lines show the update mechanisms, the programs used for the update are printed in bold face type near the update lines. The arrow heads indicate the directions of update that are permissible.

The rdist master tree represents the head revision of every file in the CVS repository[2]. The rdist master tree is updated from the CVS repository before an rdist is done. While hand editing of files in the rdist tree is possible, distributing these modified files requires circumventing the standard distribution mechanism, and thus is done in only exceptional circumstances.

There are two program components for a user of the config system:

- A Bourne Shell wrapper program for rdist version 6 that performs the underlying updates and consistency checks for the rdist master tree. This script is called capital ''R'' – Rdist.
- CVS version 1.3 is used to check modified files into the CVS repository.

In addition to the two programs above, some mechanism for allowing the Rdist script to be executed as root must be provided. We use sudo [4] version 1.3.

The programs that are used by the system itself are not that much more complex, but there are more of them.

- rdist [1] version 6.0 or newer. While config can be implemented with earlier versions of rdist, there is additional administrative overhead and performance/logging issues to consider.
- CVS is used to update the rdist master tree from the CVS repository.
- Gnu make is used to generate machine and group specific files from the sources maintained under CVS control.
- A perl program collects data from a simple database file and emits this data as a set of rdist macros (classes) for use in a distfile. This tool is called class_gen, since its original purpose was to generate classes for rdist. It has since been modified to produce `/etc/hosts` style output as well as to produce a list of host names matching various criteria.

The Distribution Mechanism, rdist and Friends

This section covers the end node distribution mechanism. This encompasses, rdist version 6, the Rdist shell wrapper, the Distfile and the arrangement of the rdist master tree.

rdist Version 6

The config system is based on Rdist version 6.0 or newer. This version of rdist has a number of features that enhances the distribution portion of the config system:

- Multiple machines can be updated in parallel. This dramatically speeds update performance.
- Set operations such as union, difference and intersection can be performed between sets of hosts. This makes it much easier to rdist files to all hosts except a few hosts.
- More information is logged. Logging using syslog is supported.
- Handling of downed or non-responsive hosts is much improved.
- It is portable to many platforms.
- With version 6.1 rdist no longer needs to be setuid root. It can also use more secure communications channels such as a Kerberized rsh.
- A single command can be run after all files in a rdist command have been distributed.
- Changed versions of files can be saved before being overwritten.

Because of rdist's parallel update capability this system scales well to a few hundred hosts. When scaling to a very large site, multiple rdist master trees on multiple hosts and a hierarchical update cascade mechanism should be used to facilitate timely updates.

The Rdist Wrapper Script

The rdist program is called from a Bourne shell script wrapper called Rdist, which performs a number of functions:

- Generates a list of directories that are to be distributed[3], and invokes ''cvs update'' on each directory to pick up the most recent

[2] Some end files are automatically generated from template files, and only the template files and the mechanism for generating the end files are kept under version control.

[3] As you will see in section ''The Distfile and the rdist master tree'' there is usually a one to one correspondence between distfile targets and the directories in the rdist master tree

changes to the config tree. It also verifies that no changes have occurred to files in the rdist master tree. If changes are found in the rdist master tree that are not in the CVS tree, the script aborts.

- Looks for makefiles in the directories that are to be distributed, if a makefile is found, it runs *make* (1) in that subdirectory.
- Determines what hosts are down (using ruptime or luptime) and removes them from the host list that it passes to rdist. This is a holdover from rdist versions before 6.0 that used to hang forever on downed hosts, but it is still a useful function.
- Creates the master Distfile on the fly from a template, and runs rdist. Optionally it passes the output of rdist through summary filters that make viewing the output much easier.

The Distfile and the rdist Master Tree

The Rdist shell script couples the distfile to the structure of the rdist master tree, so it makes sense to cover both of these items in the same section.

The distfile that is used by config has four target types. The first target type we call a ''standard'' type. A standard type target is labeled in the Distfile with a string with a lower case initial letter. Every top level subdirectory of the rdist master tree that starts with a lower case letter has a corresponding target in the distfile. Using the directory names as targets provides a nice update paradigm of:

```
% cvs co crontabs
% cvs ci crontabs
% sudo Rdist crontabs
```

```
binaries:
/config/Bindist/yppasswd.save -> (${IRIX_HOSTS} ${RISCOS_HOSTS})
      install -onumchkgroup /usr/bin/.yppasswd.save ;
      special /config/Bindist/yppasswd.save "/bin/sh \
                    /usr/bin/.yppasswd.save";
dots:
/config/dots/{.forward,.rhosts} -> ${ALL_HOSTS} - ( ${BSDI_HOSTS} ra )
      install / ;
hosts:
/config/hosts/hosts -> (${ALL_HOSTS})
      install /etc/hosts;
kernel:
/config/kernel/sinix/stune -> (${SINIX.D_HOSTS}) & (${afs_C_HOSTS})
      install /etc/conf/cf.d/stune ;
      special /config/kernel/sinix/stune "/etc/conf/bin/idbuild";
sendmail:
/config/sendmail/ida.cf/mailhost.cf -> (${MAIL_HOSTS})
     install -b /etc/sendmail.cf;
     special /config/sendmail/ida.cf/mailhost.cf
                    "/usr/local/etc/restartmail";
sendmail:
/config/sendmail/ida.cf/mailslave.cf -> (${MAIL_SLAVES})
     install -b /etc/sendmail.cf;
     special /config/sendmail/ida.cf/mailslave.cf
                    "/usr/lib/sendmail -bz";
tcpd_inst:
/config/tcpd/ultrix.bin/{tcpd.install,tcpd} -> (${ULTRIX_HOSTS})
     install -b /usr/etc ;
     special /config/tcpd/ultrix.bin/tcpd "/usr/etc/tcpd.install dec" ;
xntp:
/config/xntp/dlws9 -> dlws9.me.com
     install /etc/ntp.conf ;
xntp:
/config/xntp/dl5000 -> dl5000.me.com
     install /etc/ntp.conf ;
```

Figure 2: Sample entries in the distfile.base base distfile

When Rdist is invoked without specifying targets on the command line, it generates a list of targets for passing to rdist using `echo [a-z]*`. Thus an Rdist without any explicit targets distributes all of the standard targets. This operation is done nightly out of cron. We have come to count on the nightly Rdist cleaning up experimental versions of configuration files (e.g. inetd.conf, services) that may be changed for a short term test, but that are not supposed to be permanent changes.

Some ''standard'' targets are actually dummy targets, consisting of a Makefile that forces a CVS update of appropriate files before rdist is invoked. While use of such dummies is discouraged, there are times when they are useful when delegating responsibility for file modification.

The second type of target name is a subelement type. These appear as `<subdir>.modifier.` These cause the `<subdir>` to be updated as described above. It is policy that these subelement types distribute a subset of the files that the regular `<subdir>` target would distribute.

The third type is an install type. These are of the form `<subdir>_inst`. These cause the corresponding standard target directory (i.e., `<subdir>`) to be updated before a distribution occurs. These are reserved for single shot operations that are done when initially installing a machine. For example initial kernel config files that are expected to be modified by a later package installation, empty directory creation or wrapper program installation with special commands that are best not repeated.

The fourth type of target is called an ''action'' target. These targets are similar to the install targets in that some action is performed that may not always be necessary. One example of an action target is the `rc.reboot` target. As you might guess, this target distributes the rc files, and then forces the machine to reboot.

The Distfile is generated from the output of the class_gen script and the file distfile.base using the Makefile in the Db directory. The distfile.base is what most people think of as a distfile. Figure 2 shows some typical entries.

Actual host names are almost never put into distfile.base. Instead canonical macros are used in place of host names. These canonical macros are produced by the config_class program using the information in the database file. Config_class and its database file are described in the database section.

The Rdist Master Tree

The structure of a typical rdist tree is shown in Figure 3. The Db directory contains various administration files. As the name Db implies, there is a simple database file contained in the Db directory. Originally that was all that was in the Db directory, however as the system matured, the Db directory was used for general administration files, and file generation. Thus a better name for the Db directory would be Admin.

A perl script creates rdist macro definitions that are prepended to the base distfile to create the Distfile that is used by rdist. The Makefile in the Db directory is invoked by the Rdist wrapper script and makes certain that the Distfile is brought up to date.

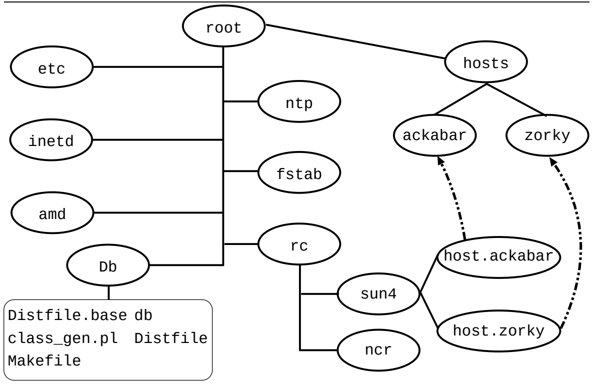

Figure 3: A typical portion of a CVS file tree repository showing some of the contents of the Db database directory.

In Figure 3, directories (i.e., standard targets) are represented by ovals. The directory ''hosts'' is special. It represents files that have been delegated to the owners of those hosts. The ''hosts'' subtree has subdirectories named after the hosts which receive delegated files. Each subdirectory has a structure that is identical to the structure of the rdist master directory, but is writable by the delegates. The separate subtree for the delegates files is required by CVS's check-in mechanism. Write permission on the repository directory is required for CVS check-in, but it is not appropriate that delegates have write permission on rdist master directories.

The segregation of files in the ''hosts'' tree produces a slight problem. When we type ''Rdist rc'', we want all rc files pushed to all hosts. However, when CVS runs down the rc directory tree to update all of the files, it will miss the files in `hosts/*/rc`. We could rewrite the Rdist script to run down the ''hosts'' tree and start CVS update in the appropriate directories, but John chose another method. Each subdirectory under the host specific tree is linked into the main tree. These links are indicated by the dashed lines in the diagram. CVS traverses the links and automatically updates the files found there. The links are automatically created by a script run on checkout for files in the ''hosts'' tree.

CVS

CVS is somewhat different from other well known version control programs in that it supports parallel development with a merge operation at check-in rather than the serial locking paradigm supported by SCCS and RCS. Due to this parallel development ability, multiple system administrators can make changes to a given file, during a crisis, without wasting time trying to find out who has the file locked. This is very important in an interrupt driven job such as system administration, since it is impossible to tell in advance just what files may need to be modified in a given day. Most configuration files are text files, or are derivable from text files, so CVS's line by line conflict resolution mechanism works very well and allows small simple changes to be done to the configuration files while other longer running changes are still in process. CVS's conflict resolution mechanism does not work well on binary files. In our experience binaries files change infrequently. Changing binaries is not a ''quick fix'' type of operation, so the delay inherent in a serial locking paradigm is not a problem. We use the hooks in CVS to provide a serial locking paradigm when changing binary files.

CVS actually uses RCS internally, and thus provides most of the functionality of RCS such as named version information (useful for making checkpoints on the first of the month). The more mundane, but crucial aspects of a version control system are also provided. Namely, all previous versions of the files are available from the RCS configuration files and the names of the people who made changes are recorded for posterity.

One additional and crucial advantage of using CVS as opposed to RCS is its handling of log information. The log information can be routed to people through various means. News and email have been the standard mechanism up to now, but one site where config is deployed has been using alphanumeric pagers and the inform system for real time notification of changes to the CVS tree. One additional feature of the the CVS log facility is that different people can be notified about changes in different parts of the tree. So a change made to files for the host ackbar, can trigger email to the person who is supposed to be maintaining that system.

Recently CVS's ability to run verification programs on check-in has been exploited. If any of these verification programs exit with a non-zero status, the check in is aborted. These programs are being used to provide sanity checks on data files, check to see if particular files are locked, or make sure that the person attempting to make the change has permission. For instance the amdxref program was used to validate amd automounter maps during check-in to reduce the chances of an incorrect map being checked in. Sanity checks for hosts files, and inetd.conf files have also been implemented. Also `sh -n` makes a pretty good sanity check for shell scripts such as rc files.

Gnu Make

We use GNU make in preference to other makes because it provides some very useful features (e.g., automatic makefile generation, and tracing ability). Also it is available across a large number of platforms making the rdist master tree platform independent. This allows rdist master trees to be scattered across multiple platforms which is important for scalability, and may be required for availability. While the use of make is not required, it provides each standard target with its own method of generating host or class specific files. These methods can be as trivial or complex as needed. These methods ease the task of maintenance especially when there are only minor differences among the various files.

As part of the Rdist wrapper, each top level directory is checked for a makefile. If one is found, a make is done in that directory. These inferior makes can be used for a number of purposes. One purpose is to generate host specific files such as `inetd.conf`, or `crontab`. Using cpp and sed, different version of the files can be generated from a generic template file. These generated files are distributed via rdist. Using prototype files in this way greatly reduces the amount of work that must be done when performing a global change such as installing a new housekeeping program, or adding a new experimental service.

The Database System

A simple database is used to record essential information about the hosts handled by the config system. This information includes: the machine name, the machine's operating system and version information, its administrative group and the services it provides and uses. A perl script, called class_gen, creates rdist macro definitions that are prepended to the base distfile. This composite Distfile is used by rdist. The Makefile in the Db directory is invoked by the Rdist wrapper script to make certain that the Distfile is kept up to date.

The Database File

Figure 4 shows two sample database entries. The format of the entries in the database file is a simple `keyword = value` syntax. A host entry starts with the ''Machine'' keyword and continues until the next ''Machine'' keyword. The keywords: cluster, os, patches, serves and uses are used by the class_gen program to create classes of host for rdist. To ensure that the data in the database is up to date, simple scripts have been written that verify information such as the os and os revision and the amount of disk and memory.

When first installing a machine, the appropriate information is set up by hand. A special version of the Rdist script, called Install, is used to bring the newly installed machine up to date with the master rdist tree.

```
Machine = dl5000.me.com
Alias = dl5000 nexus master
arch = D5000/200
cluster = bedford
ip=132.121.14.10
enet= 00:00:08:04:05
os = Ultrix 4.2
serves = jukebox decnet bind time
serves = mail pci lat print_lp
serves = syslogm fingerm
uses = network
patches = 002-34A 005-153C 82352-bc
pmemory = 48M
kernel_name = nexus
```

and

```
Machine = amethyst
alias = am
ip = 136.121.32.20
cluster = dev
arch = R4000
cpu = R4000
pmemory = 32M
disk = 300M
owner = gink
# os = type  and version
# with .'s between
# revision steps.
os = IRIX 4.0.5.f
```

Figure 4: Two sample database entries.

```
ALL_HOSTS=( amethyst chicago claude
   god iris orchid ra violet )

IRIX_4.0.5=( iris violet )
IRIX_4.0.5.F=( amethyst )
IRIX_4.0.5.X=( amethyst )
IRIX_4.0.X=( amethyst iris violet)
IRIX_4.X=( amethyst iris violet )
IRIX_5.0=( orchid )
IRIX_5.X=( orchid )
IRIX_HOSTS=( amethyst iris orchid
    violet )
DOS_3.1=( god )
DOS_3.X=( god )
DOS_HOSTS=( chicago god )

es_C_HOSTS=( ra claude iris)

SYSLOGM_HOSTS=( ra )
BIND_HOSTS=( ra claude )
```

Figure 5: Representative output from the class_gen program when used to generate rdist macros.

Class_gen

Sample output from class_gen is shown in Figure 5 and includes four types of output:

- Global output. This is shown by the ALL_HOSTS macro.
- Host os/version. Multiple macros are created for each os/version item in the database file. There is one macro created for every os version, and every os parent version. This cascade can be seen with IRIX hosts. Not to be left out, DOS as well as all other host types can be represented.
- Cluster groups. It is often useful to be able to group machine by task, or by the workgroup to which they are assigned. In figure ''Class_gen'' the Engineering services cluster hosts are listed with the canonical name `es_C_HOSTS`.
- Hosts that serve particular information for example they are syslog master servers, or supply bind (DNS) service have their own groups.

In addition to the macros shown in Figure 5, the database keywords: ''uses'' and ''patches'' produce their own classes in much the same way that the ''serves'' keyword produces classes.

Tasks

The config system gets a host into a known state automatically and repeatably. This ability eases a number of tasks.

Installing/Upgrading a System/Crash Recovery

The process of installing a new system, of a type already handled in the config system, starts with a stock os installation. After the installation has been done, and the machine is on the network, add an entry to the database file, and run rdist against the machine until no more files are transferred. Then reboot the machine and it should be fully configured.

Crash recovery is also greatly simplified since there is no need to go to backup tapes to reload the base machine configuration after installing the base system.

Updating the operating system is handled the same as installation because we don't trust the ''Fast Upgrade'' procedures that vendors come out with. Also, an upgrade is a good excuse to clear out all of the cruft from the filesystem.

The Update Procedure

Updating a file under CVS control is very easy.

- Check out your own local copy of the file in question using ''cvs co''.
- Edit the file appropriately.
- Perform a test install of the file to make sure that it works as expected. [optional but suggested].
- Check in the new file using ''cvs ci''.
- Distribute the file using Rdist. If you didn't perform a test install, it is advisable to rdist to a subset of hosts for testing purposes before distributing to the whole network.

We find that people are usually very good about testing their changes before inflicting them on the world since their name is associated with the check in notice that is distributed via email, and news. If they aren't good about it, one or two mistakes later, they are very good about it. It is recommended that sanity checking programs be crafted for the files since using this feature makes it much safer to have novices changing files. At the very least the sanity check programs will prevent catastrophic errors from being checked in. When you have novices changing files, sanity checking programs[4] can be considered mandatory.

Creating New Targets

When creating new targets, the most difficult question is where to put them in the config tree hierarchy. Targets that are likely to be changed often should be near the top of the tree to make it easy to remember the target names, and to make the target name (or CVS module/directory name) easy and fast to type. It also makes it easier for new system administrators to develop a feeling for the most heavily used files, and thus the most active tasks.

In addition to structuring your tree by function (e.g. directories for passwords, patches, etc) it is often useful to create directories that mimic the tree structure where the files would be found. For example the /etc/services file has the same structure on all hosts. It is usually an easy matter to produce a single services file that will satisfy all of the hosts in the network. Files like these can often be placed into a directory such as ''etc''.

Some files such as fstab/vfstab files aren't often changed, but it seems that each architecture has its own variation on the contents of these files. A reasonable location to put these files would be in etc/fstab, and populate the directory with the vfstabs or fstabs for the different architectures, or machines. A subelement target (e.g., etc.fstab) could be used to force distribution of just the fstab/vfstab files.

We have found that the subelement targets are rarely used in practice. In most cases, the encompassing simple target is used instead. The only exceptions to this are for those simple targets that use lots of data. For example, the target ''Patches.solaris'' exists in both the CVS module definition files and in the distfile. This reduces the data that must be checked out of the Patch directory when updating the solaris patch tree. There is no need to maintain a one to one correspondence between CVS modules and the subelement targets in the distfile, although such a coupling can be made. The implicit relation that the non-standard targets have to their standard targets provides the coupling into the CVS config tree; it is only at this level that the CVS tree and the distfiles are tightly coupled.

One additional feature of config, is that you don't have to start using all of the features of config at once. When creating a target, you can start out with a simple target/directory that has one file for each host enrolled in config. This would produce a large distfile, but it would be simple to understand, and very little could go wrong.

As you tire of making the same changes to multiple copies of the same file, the various classes that are created using the database file, and the class_gen script can reduce the needed changes to files that actually have different contents. If even this reduction in work is not sufficient, the ability of config to run a makefile to generate the individual class files from a template can be exploited. Autogeneration of files can fail, incorrect database entries, disk full conditions, etc. can interfere with the proper operation of the system, but we have found that these failure modes occur infrequently and are easily diagnosed and fixed.

Delegating Responsibility For Files

When you have to delegate responsibility for files there are a number of methods by which access can be granted. The easiest is to use RCS's own access list policy to restrict the people who can access a file. Alternatively or in addition, other access checks can be performed by the programs that are run upon a CVS check-in. Once access is granted via whatever means, and suitable sanity checking programs are put in place, it is very easy to allow people to change files.

However, it is critically important that the master distfile.base is kept under tight control. This way the person is unable to ''delegate'' himself responsibility for extra files, since these extra files won't be propagated to the end node machines.

Also, it is important to prevent the novice system administrator from running anything but ''standard'' rdist targets since install and action targets can cause actions to occur that would be detrimental to host operation.

Fixing Problems with Hosts

When a problem is encountered with a host, the first step is to run Rdist against that host to make sure that it has all of the proper files in place. In the past many problems have been handled simply by distributing the known configuration files onto the remote host, possibly followed by a reboot.

Sadly not all problems can be solved so easily. This is where the CVS component of the system shines. The question that is invariably asked is: ''What has changed since *date* when everything was working?''. There are a couple of ways of answering this question. If you have been using CVS's logging ability to its fullest, there is a newsgroup, or a mail file folder that has a list of all logs that have been produced. Then is it a simple matter of looking at the logs after a particular date.

[4] For the files, if not the people.

Another way to answer the question is to use the rdist master tree itself. A cd to the top of the rdist master tree followed by a ''cvs log -d `date`'' command shows all of the log information on files checked in since the specified date. Sadly, the RCS rlog command also lists header information for files that do not have new revisions in the date range specified, but this is easily handled using a wrapper script that filters out the unneeded information.

One additional function that was not originally envisioned was to link the CVS log reports into a trouble reporting system by using a site specific script to filter the log. Using this method, changes recorded under CVS can be automatically entered into a trouble reporting system, and can be linked to the corresponding trouble ticket. This provides an easy mechanism for formally documenting the changes necessary for a problem's solution.

With change information in hand, the job of analyzing the cause of the problem is often greatly simplified.

File Scanning

It is possible that a change is made to files that are not under the config system. In this case, the host can not be recreated from the os installation tapes, and the information in the rdist area. Troubleshooting problems under these circumstances is difficult if the cause of the malfunction is this uncatalogued file.

A system integrity checker is used to discover these unenrolled files. Since we use tripwire to check system integrity for security purposes, it was only natural that it also would be used for checking the integrity of the entire system. For the purpose of insuring integrity for config, a simple checksum such as crc-16 or crc-32 can be used saving a lot of time that is needed to compute more sophisticated checksums such as MD4 or Snefru. We have had good luck with using tripwire in this ''simple'' mode to discover files that should have been put under config control, but weren't.

Conclusion

This system has been used to provide support from networks of 10 machines with two administrators to 280 machines with 10 administrators. We have found the above system to be very flexible and easily extendable to support many different notification mechanisms. Using multiple rdist master trees on different hosts, and keeping a duplicate copy of the CVS master tree on a second rdist master host allows the system to be quite robust in the face of network interruptions and system downtime.

However all is not perfect. There are a few areas that we will be concentrating on for future development. As the system currently stands, it is possible for config users to circumvent the logging mechanism by directly modifying the CVS version control files. This problem can be minimized or potentially eliminated by running CVS and the underlying RCS programs setuid. In practice we have found this to not be a problem since updating and distributing files using the standard mechanism is much easier than trying to change the CVS repository files by hand, however some provision for supporting hostile users must be made.

A second problem stems from the system's flexibility. Because the system is so flexible, it is easy to create a config tree structure that makes it difficult to find the files that you want to change. RCS $source:$ strings mitigate the problem, but they are not definitive: automounting and symbolic links can confuse the issue. Also as the config tree evolves, CVS's lack of support for renaming complicates rearrangement due to paradigm shifts.

Availability

The class_gen perl script and the slides for this talk will be available from anonymous ftp site ftp.cs.umb.edu in the files:

/pub/bblisa/talks/config/config.tar.Z
/pub/bblisa/talks/config/config.slides.tar.Z

Rdist version 6.x is available from usc.edu, CVS version 1.3 is available from UUNET and was submitted to comp.sources. The master Rdist script is available but note that it is so crufty, that it is getting unmaintainable, and needs a complete rewrite.

Author Information

John Rouillard graduated from the University of Massachusetts at Boston with a B.S. in Physics in 1990. Since then he has been a contractor specializing in tool building and automation of various system administration tasks. In January of 1994 he took over release engineering and development from Brent Chapman for the Majordomo mailing list management tool. At the same time, he became a Senior Systems Consultant for the Mathematics and Computer Sciences Department's Software Engineering and Research Labs at the Univ. of Massachusetts at Boston where he continues his system administration automation tasks for various Lab clients.

Richard Martin attended Merrimack College and UMass-Boston. He has been System Programmer with the department of Math and Computer Science since 1984, maintaining hardware and software and training system administrators.

Bibliography

[1] Cooper, Michael. ''Overhauling Rdist for the '90s'', LISA VI proceedings, October 1992, pp 175-188.

[2] Jones, George M. and Steven M. Romig, ''Cloning Customized Hosts (or Customizing Cloned Hosts)'', LISA V proceedings, September 1991, pp 233-241.

[3] Kint, Richard W. ''SCRAPE (System Configuration Resource and Process Exception) Monitor'', LISA V proceedings, September 1991, pp 217-226.

[4] Nemeth, Evi and Garth Snyder and Scott Seebass, ''UNIX System Administration Handbook'', Prentice Hall, 1989.

[5] Rich, Kenneth and Scott Leadley. ''*hobgoblin*: A File and Directory Auditor'', LISA V proceedings, September 1991, pp 199-207.

[6] Rosenstein, Mark and Ezra Peisach, ''Mkserv – Workstation Customization and Privatization'', LISA VI proceedings, October 1992, pp 89-95.

[7] Zwicky, Elizabeth D., ''Typecast: Beyond Cloned Hosts'', LISA VI proceedings, October 1992, pp 73-78