Using CVS to Administer Linux

Horms (Simon Horman)
horms@zipworld.com.au.

January 1999

http://linux.zipworld.com.au./

"Alles hat eine ende, aber ein Wurst hat zwie." [1]

–Unknown

[1] Everything has an ending but, a sausage has two.

Acknowledgements

As always I would like to acknowledge the generous support of my employer Zip World without whose help this paper would not have been possible. Special thanks goes to BB for helping me to get the bits on routers correct.

Version Information

Version	Date	Description
0.0.0	22nd, January 1999	Meat Pies and Tomato Sauce Prerelease. Conclusion is missing.
1.0.0	1st, February 1999	Woomera Release. Released to Linux World Expo.
1.0.1	4th, February 1999	Twinkle-Toes Release. Minor Corrections.

Presented

Canberra Linux Users Group (CLUG)
28th January 1999
Australian National University
Canberra
Australian Capital Territory
Australia

Linux World Expo
2nd-4th March 1999
San Jose Convention Center
San Jose
California
United States of America

Abstract

Much emphasis has of late been put on making Linux easier to manage. Typically the model used it that of a single user managing one or more systems. The environment I live in is not so simple. Multiple administrators looking after the same set of systems. To add to the complexity of the situation a myriad of scripts are in operation that are accessed by the system administrators, other trusted users and even via web pages after user authentication.

Naturally this sort of situation causes headaches. There needs to be some mechanism to arbitrate access to configuration files. Additionally a means of recovering old versions is particularly useful in the event of bad changes made inadvertently by someone or by a rouge script.

This problem is in essence very similar to the problem software developers face when a team is working on a source tree. There are many solutions to this problem, one of the most popular being Concurrent Versions System or CVS. By placing the configuration of a Linux system under CVS we are able to merge changes made by multiple users, recover old configurations and checkout configurations into new environments. Constructing multiple environments for development, test and production is also simplified. Additionally the CVS repository acts as an backup mechanism that can be used to supplement existing procedures to guard against catastrophe.

The problem is particularly interesting as it in clearly not just a matter of constructing a technically correct solution. This is a system that by its very nature is used by many people on a day to day basis. As such the system must be easy to use and where possible require no modification of existing processes.

Contents

List of Figures

1 Introduction

In administering Linux Servers for an Internet Services Provider it has become necessary for configuration files to be modified by multiple users and scripts. This raises several serious issues that need to be address to ensure the integrity of the system.

1. **Concurrent Access**. Some mediation is required to ensure that all simultaneous edits to files are accepted.
2. **Roll Back**. A mechanism is required such that either user or script errors can readily be reversed.
3. **Audit Trail**. Edits need to be traceable to a user or script in order to prevent the recurrence of problems that may occur from time to time.

Roll back and audit trails are essentially version management issues and a system such as The Revision Control System, RCS or Source Code Control System, SCCS can be employed. Systems such as RCS and SCCS do not, however, provide for concurrent access. Concurrent Versions System, CVS is a revision control system, built on RCS that allows files to be checked out concurrently by multiple users. CVS manages edits of the same file by multiple users transparently unless the same part of the same file is edited in which case manual conflict resolution is required. As management of concurrent access is a key issue that needs to be addressed CVS was chosen as the basis of the system as it provides all three attributes that we require.

2 Basic Architecture

The initial motivation was to create a system that would require minimal changes to scripts and administrative processes. For this reason the system developed has configuration file edited in place on the production system. At the conclusion of an edit by an administrator or script make is run. The Makefile written makes CVS commits as required and performs any other

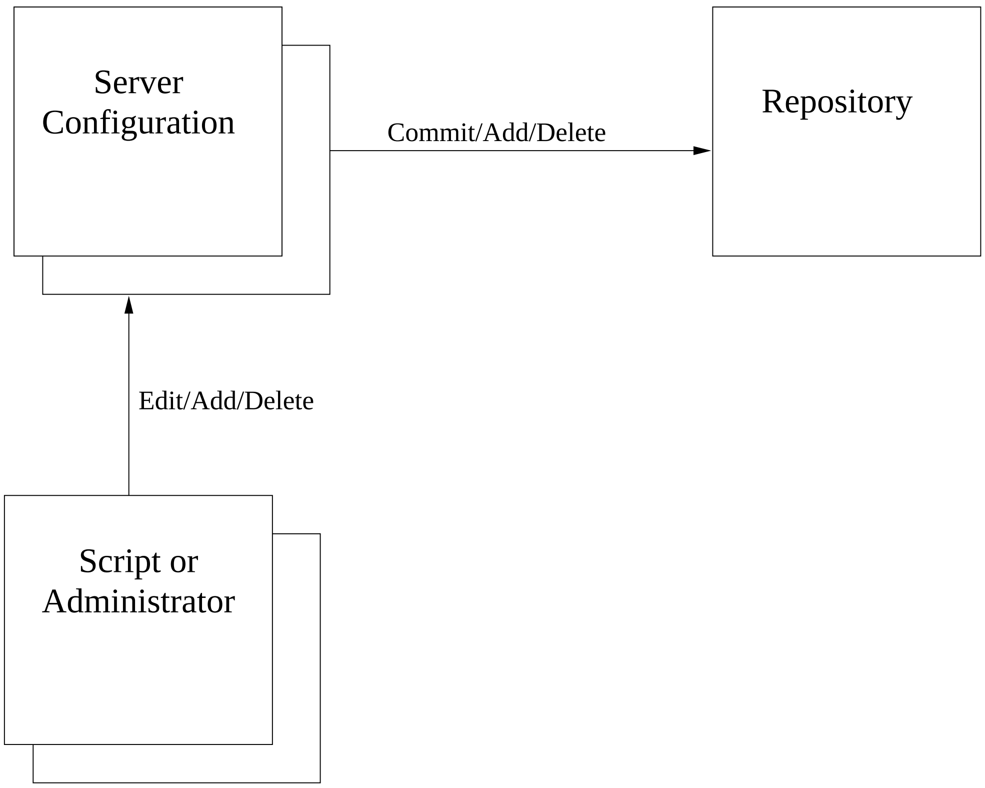

Figure 1: Basic Architecture

tasks required to effect the change. Such tasks include the conversion of plain text files to DBM files and sending signals to daemons as appropriate.

Addition and removal of files from the repository is dealt with by scripts that create or delete configuration files. An example of files that may be added and deleted are zone files or email configuration files for virtual domains. The DNS and email servers that this system was designed to work with have many such domains and it is inevitable that domains will frequently be added or deleted.

The issue of concurrent edits is primitively handled by the use of vim style lock files. Scripts that with so access configuration files honor this lock file semantic so it is not possible for more than one script or administrator to access a file simultaneously. This also requires all administrators to use vim.

A symbolic link from vi to vim and an absence of administrators who use editors other than vi have worked well to date. This locking mechanism while extremely simple to break eliminates most fine contention issues.

In this system CVS provides a rollback mechanism as arbitrary revisions of files in the repository can be constructed using CVS diff. This is ideally suited to situations where configuration files mysteriously stop functioning invariably over the course of a long weekend where no one recalls having made any changes.

The audit trail afforded by this system is limited as all commits come from a single user on a single host however the time of commits is recorded and this is certainly an improvement on no audit trail at all.

3 Version Control on Routers

As equipment such as routers are generally unable to have version control software such as CVS installed it is necessary to slightly modify the system to archive router configurations. This is achieved by downloading the configurations to a box that has CVS installed and then adding committing and deleting files in the repository from there. The download of files can be done either as changes are made or periodically depending on the frequency of changes, the number of administrators that change router configurations and personal preference.

In the case of a Cisco router the download of configuration files can be effected be either of the following two commands after enabling rsh access from the machine that you are downloading the files to.

```
rsh <router-name> sh run > <router-name>

rcp <router-name>:running-config <router-name>
```

This system relies on the router to provide concurrency control of access to the configuration which in the case of routers I have had experience with

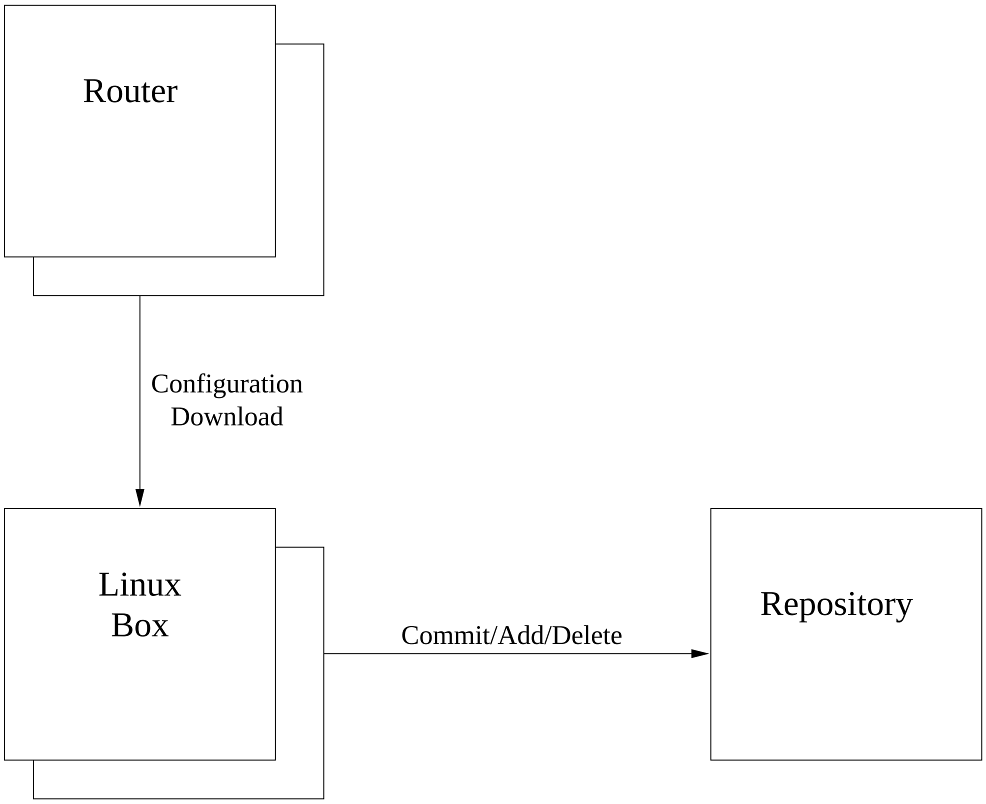

Figure 2: Architecture Used for Routers

is essentially no concurrency mediation to speak of. The roll back and audit trail semantics are the same as for the original architecture though the granularity of this depends on the frequency of the download and commit cycle.

4 Extended Architecture

The lock file concurrency control that is implemented for the architectures covers so far does not scale well. An administrator can quite easily hold a file open for longer that it is reasonable for a script to try and gain access, especially if these scripts are interfaced off the web and are expected to run

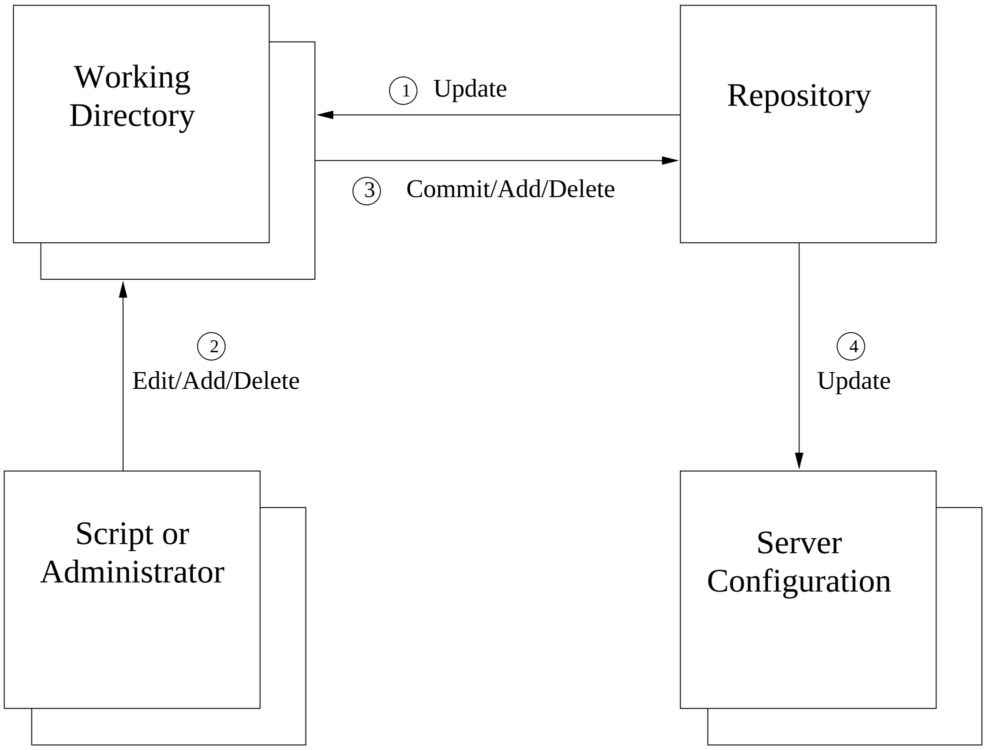

Figure 3: Extended Architecture

in real time. An additional short coming of the systems outlines is that edits are made directly to the production system. This can be dangerous if partial edits are written to disk and the system is then committed by some other process bringing those changes into effect.

An alternative, though more complicated system involves each user and each script having their own working directory in which their edits are made. This fits more closely to the way that CVS is intended to be used and as a result the concurrency control build into CVS can be brought into play.

Each user or script checks out the configuration files into a local directory that is local to them. Checkouts are expensive but after the first time a CVS update can be used to ensure that the local tree is up to date. Updates are quite cheap in comparison to checkouts. Edits are made to the configuration and once these are completed a commit is made placing the changes made

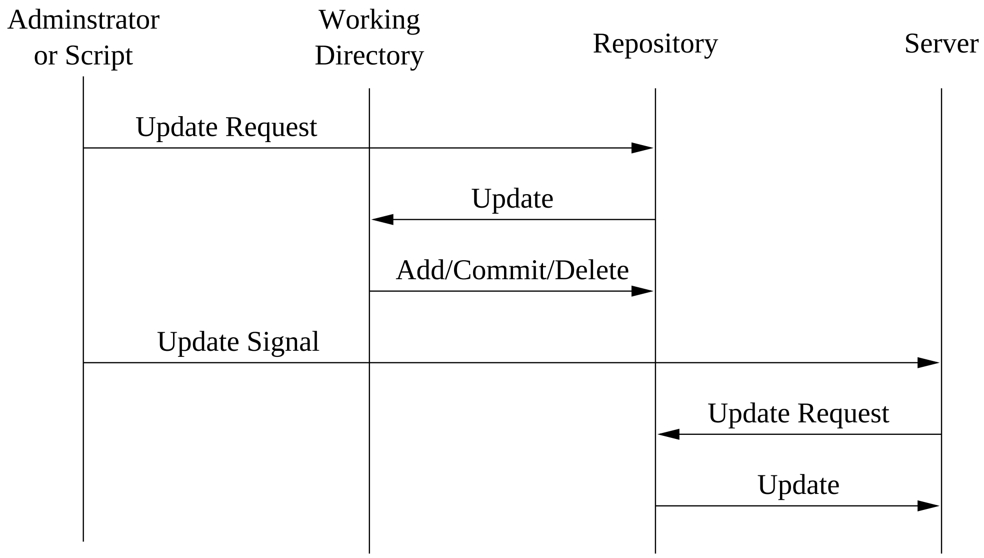

Figure 4: Extended Architecture - Entity Relationships

into the repository. Finally an update is run on the production system in order for the changes to become 'live'.

In the case where there is contention for the edit of a file that CVS cannot resolve and a commit is being made by a script this can be considered to be an error state. When this occurs some form of notification should be given to enable human intervention and any changes made to the local directory should be rolled back.

These processes can once again be automated with the aid of make and even the updated on the production system can be automated being triggered either by make in the local directory or run periodically as a cron job.

Aside from giving greater concurrency control this system also offers the potential for a stronger audit trail as the commits made by different users and scripts can be more easily traced, depending on the access mechanism.

5 Security Issues

Ideally archives should be on a remote machine. This has a number of advantages including: insurance against disk failure on the server; the ability to have a central repository for all configurations; using the extended architecture, the ability of users and scripts to edit configuration file(s) ? off the server.

If a remote archive is to be used then critical server configurations are to be passed across a network and it is crucial that this be done in a secure fashion. There are a number of different methods of accessing CVS remotely with varying degrees of functionality and security attached.

The default remote access mechanism for CVS, pserver, provides a high level of functionality as it is session based and once authenticated operations can be performed without the need for additional authentication. Unfortunately in its native state this could be well described as insecure. This is not a problem for anonymous CVS servers or a CVS repository in a secure environment. The system developed here is, however, intended to be run in the hostile environment that becomes servers at an ISP.

The port redirection functionality of SSH can be employed to provide an encrypted pipe across which CVS can operate. Alternatively ppp can be run over ssh to provide a secure tunnel between servers and the repository. Both of these solutions are reasonably high maintenance as the ssh tunnel or pipe needs to be maintained and depending on firewall restrictions an insecure connection may be possible and undetected in the case where the ssh link goes down as is its wont.

The preferred solution to my mind is to use ssh as the transport agent for CVS. This does require authentication for each transaction resulting in increased overhead. This can be minimised by ensuring that transactions such as commits and adds are grouped than initiating a separate transaction for each file. This is more an implementation issue rather than a technical barrier. Also RSA keys with null pass phrases can be used to enable secure authentication without the need to enter passwords or pass phrases.

Finally mechanisms such as Kerberos and the SSL library can be employed to provide secure connections to the repository. I have had very little experience with these methods and will not comment any further on their use.

6 Conclusion

The key is to create a mechanism that enhances data integrity while minimising the alterations required to the way that people work. Roll back and audit mechanisms can be provided using traditional revision control mechanisms such as RCS and SCCS. These do not provide for concurrent access to files, something which CVS does provide. To this end CVS has been chosen as the basis for the system.

An easy first step is to allow files to be edited in place and then commit changes to a CVS repository. While this does not satisfactorily address concurrency it does provide for roll back and a limited audit trail, as well as requiring a minimal amount of change in procedures.

A more far-reaching approach is to have users check out the repository and edit it locally. Changes are then committed and make their way into production using CVS update. This is a more robust solution as CVS provides concurrency control.

As CVS is likely to be used in a networked environment, it is important that security issues are kept in mind. The standard remote access mechanism for CVS is not secure but tools such as ssh enable transactions to be conducted in a secure manner.

The importance of the additional data integrity that this system can provide is highlighted by a phone call I received from a colleague while on the way to Canberra to present this paper for the first time. A configuration file had been truncated and recovery from tape would lose the day's edits. As this file was under CVS control it was a simple matter to recover the most recent revision and put this back into production. While this shows how effective this system can be it also draws attention for the need for accessibility. While the day to day commits required have been made accessible through transparency, recovery mechanisms also need to be made available.

To gain increased reliability it is as important for the knowledge of how to use a system to be freely available as it is to provide redundant data.

A Terms

- **DBM**: File format that contains hashed key value pairs.
- **cron job**: From Chronological Job: A scheduled command that is run periodically at a set time.
- **Kerberos**: System for making secure connections across a network.
- **make**: Utility for automatically executing commands based on the age of files. Typically used to maintain compiled versions of source code but can readily be used to maintain any system where files can be viewed to be dependent on the revision of other files.
- **vi**: The generic text editor for Unix.
- **vim**: "Vi IMproved, a programmers text editor". An extended version of the vi editor that amongst other things creates lock files for files that are being edited.
- **ssh**: Secure SHell. An suite that allows for encrypted shell and other data transfer between hosts.
- **SSL**: Secure Sockets Layer. Library that provides authentication and encryption for data transfer.