A Software engineering Approach to Computer Networks Management

Author: Gangadhar Chalapaka

Supervisor: Hans Bjurgren

Co-Supervisor & Examiner: Sasikumar Punnekkat

October 19, 2009
Abstract

The main objective of this thesis work is to study the problems in maintaining the computer networks and come up with a system which resolves the issues present in modern networking environments. The objective of this project is to adopt the software engineering approach for the efficient design, implementation and maintenance of computer networks and systems. Besides this the enterprise level network managing products which offer a great level of flexibility and ease of use, are at a premium. The outcome of the present study may give an adaptable, cost effective system for organizations.

Employing software engineering principles in software development process gives a systematic, disciplined approach to the development, better operation and maintenance of software products. Software engineering principles are extensively used for application development, but seldom employed for design and implementation of networks. Software engineering concepts like versioning, modeling, documenting and testing can be effectively used for the design and maintenance of networks and systems as well. The main problem in maintaining the systems data comes in absence of documentation.

The projects goal is to come up with a ”build” system and an integrated process that dynamically generates configuration files and scripts. These correctly generated, interdependent scripts ensure the functionality and reliability of the integrated networks.
# Contents

1 Introduction ................................................. 1
   1.1 The scenario .............................................. 2
   1.2 Software engineering .................................... 3

2 Related work ................................................. 5
   2.1 Cfengine .................................................... 5
   2.2 Webmin ..................................................... 6
   2.3 Firewall builder ........................................... 7
   2.4 Puppet ..................................................... 7

3 Problem statement ........................................... 8

4 Methodology .................................................. 10
   4.1 Overview .................................................. 10
   4.2 Model ..................................................... 11
   4.3 Version control ........................................... 12
      4.3.1 CVS .................................................. 13
      4.3.2 subversion ........................................... 13
   4.4 Documentation ............................................ 14
   4.5 Architecture and Implementation ....................... 15
   4.6 Building the configurations ............................ 17
      4.6.1 Creating the local build ............................ 17
   4.7 Checking host system using SNMP and deployment .... 19
      4.7.1 SNMP check .......................................... 19
      4.7.2 Deployment of configuration scripts ................. 21
      4.7.3 An example .......................................... 22
   4.8 Systems extensibility ..................................... 24
      4.8.1 SNMP integration ..................................... 24
List of Figures

4.1 Approach stage by stage ........................................ 10
4.2 Modeled Systems Data ........................................... 11
4.3 Task Management Cycle .......................................... 18
4.4 Error message: Not enough arguments .......................... 20
4.5 Error message: Invalid Host ..................................... 20
4.6 Error message: Timeout ........................................... 21
4.7 Successful output of snmp-test ................................. 21
4.8 An example cycle of operations for a host ................... 22
4.9 Backup cycle ....................................................... 24
4.10 Subversion Repository Structure ............................... 26

B.1 Directory structure of configuration ......................... 41
B.2 Build directory .................................................... 42
Preface

I express my sincere thanks to Prof. Sasikumar Punnekkat for selecting me to be a part of the EURECA (European Research and Educational Collaboration with Asia) program funded by the Erasmus Mundus External Cooperation Window (EMECW) of the European Commission, to do my project at Malardalen University, Sweden.

I was delighted to interact with Prof. Sasikumar Punnekkat and having him as my Instructor for my thesis study.

I would like to thank my Supervisor Hans Bjurgren for all his help and support for the thesis. I would also like to express my gratitude to Conny Collander and the Netcenter Team at the University for their immense help and assistance in completing the project.

I express my sincere thanks to Dr. Venkatesh Choppella, Associate Professor in Indian Institute of Information Technology and Management-Kerala, Thiruvananthapuram, for his guidance, help and support throughout my Thesis work.

My heartfelt thanks to Dr. Elizabeth Sherly, Director of IIITM-K, Kerala, whose advises and lectures helped us a lot in our internship. I would also wish to thank the systems and networking team of IIITM-K, Kerala, for their support and help.

I am grateful to all the faculty members and colleagues of IIITM-K, Kerala, for their encouragement and appreciation.

Above all, I would like to thank my parents for their continuous encouragement and support for my studies.
Chapter 1

Introduction

With rapid growth of Internet over the last few decades enormous changes occurred in computer network infrastructure all over the world, with the maintenance part of networks lagging behind. Software being developed at organizations shifted from desktop applications to web based applications. On-line services such as banking, e-learning, shopping, education portals, more than 60 % of the software applications of the world running over the web, thus the usage of web became an indispensable part of day-to-day activities. To keep the top layer of Internet or intranet more reliable the underlying layer of networking should be taken care well. Sometimes the applications running over the network may cause severe errors due to poor or erroneous or overloaded network layer.

As the network grows unpredictably, the problems associated with the back end system which provides the web interface also grows to a great extent. The number of networked computers in every organization also increased to accommodate the need for web services, thus the complexity associated with the system increases. Besides the efficient use of resources and service optimization, security issues should also be taken care to ensure reliable operation. All the organizations standards has been raised to cope up with these changes in technology. All though certain corporate companies took the advantage of the growth of Internet, developing countries are still using the open source software. Buying the big tickets from enterprise always a question for them.

Network expansion is important and inevitable. New routes, domains and services have to be added to the network to suit the organizational requirements. But as the network complexity increases, with it comes various problems in maintenance such as latencies, security and performance related issues.
1.1 The scenario

In this section we will talk about the system administration with Unix/Linux based systems. Since GNU/Linux based operating systems are open source and most of the organizations like IIITM-K, in developing countries like India showing great interest towards promoting open source software we use Gnu/Linux system as our working platform.

The Unix/Linux systems gained more popularity and best known for use in building the network infrastructure. In a typical Unix/Linux machine each system related task can be controlled by a configuration file. Any change required to a particular service offered by a system can be altered by editing the configuration file for the respective service. Operations on these files and changes to services can be made in batch mode via shell scripts.

Today, most of these configuration changes must be done manually by systems administrators. There is no standard method for performing dynamic reconfiguration of services or network elements in response to changes in the network. Script-based solutions are usually highly dependent on network topology and service element configuration mechanisms which differ across vendors and even between different versions of the same product. Scripts are frequently highly customized and often written in an ad-hoc manner. Moreover, a single change in the network can require changes in multiple scripts worsening the maintenance overhead. This has an adverse impact on the network reliability, since it is often possible that not all scripts are updated in a consistent manner. Errors in scripts can result in inconsistent network configuration states, and manual recovery made difficult by lack of logging. Moreover, each script must carefully be examined before it can be run.

These system administration tasks can be much more complicated when there are more number of systems and each of the system running more than one service. The tasks are boring, routine, critical and must do tasks. Similarly the scripts on the machines works as gateways and routers will be much more complicated and need to be verified.

The traditional way for performing these tasks is by writing scripts manually and inspecting it carefully. There is no predefined, systematic, disciplined, reliable modular paradigm for maintaining these configurations handling the routine tasks of backing up these configuration files. Testing these scripts are performed on a “trail and error” or rather “guess and check” procedure. Perhaps the initial stage of setting up a network gives good results with the above procedures, but as the network starts growing the tasks become cumbersome to handle, and the number of configuration files become unmanageable by the administrators. All the scripts need to
be backed up in order to roll back to the previous states of network in case of an error. One of the simplest way to keep the files is to save the configuration files with a time stamp in its name and with different file extensions. But this procedure is more likely to become confusing soon. As usually there is no proper documentation on what administrators do for custom tasks and scripts, it is very difficult to trace the errors or tracking the changes. Forecasting the future requirements of the organization may solve this problem up to some extent. But changes in technologies and need for new services are unpredictable. Hence, this solution can be applied only for hardware changes. A better solution is to adopt a system which will grow along with the network and accommodate the changes in system configuration.

1.2 Software engineering

One of the reasons for above problems is that these systems require that standard engineering techniques be extended to deal with new levels of complexity, new types of failure modes, and new types of problems arising in the networking environments also. Software engineering principles are used to develop higher quality of software. For example, it is well known in software engineering that the absence of Documentation for any software system will leads to difficulty in repair and recode. However organizations do not treat scripts as “software” and hence do not follow software engineering practices.

The phases in software engineering are listed below:

- Software requirements: The elicitation, analysis, specification, and validation of requirements for software.
- Software design: The design of software
- Software development
- Software testing
- Software maintenance
- Software configuration management:
- Software quality
Many of the organizations proved that the software engineering practices gives a better and reliable way of software development. Hence adopting certain software engineering phases in developing networking infrastructure also yields a reliable architecture.

For example the following are the phases can be applied to networking environments to design a reliable and better architecture for network management systems.

1. Software configuration management: Since software systems are very complex, their configuration (such as versioning and source control) have to be managed in a standardized and structured method.

2. Software development: The construction of software through the use of programming languages.

3. Software testing: Testing software products in several stages gives a error free products.

4. Software maintenance: Software systems often have problems and need enhancements for a long time after they are first completed.

The above phases imposes certain rules and regulations on the software product development, and creates a life cycle called software development life cycle. These rules makes the development to follow a systematic and organized way.

Introducing programming concepts and programming languages in networking management elements design gives us a feature software re-usability. Software data abstraction allows handling data in meaningful ways. For example, it is the basic motivation behind datatypes. Data abstraction enforces a clear separation between the abstract properties of a data and the concrete details of its implementation.

In the coming sections we will discuss how we can make use of software engineering principles and design an architecture for managing and handling the data in a systematic way.
Related work

In the following sections we briefly describe the related work. A lot of work being done towards improving the architecture, monitoring and automation of network management systems.

2.1 Cfengine

Cfengine [5] is a policy-based configuration management system written by Mark Burgess at oslo university college. The aim was to absorb frequently used coding paradigms into a declarative, domain-specific language that would offer self-documenting configuration. Its primary function is to provide automated configuration and maintenance of computers, from a policy specification. The cfengine also called as host configuration management system and works based on the policies written by the administrator. The Integration of SNMP and cfengine called SLCI [9] has been published to control the behavior of network devices by policies. The functionality of cfengine can be summarized by the following list:

- Testing and configuration of network interface
- Simple automated test file editing
- Symbolic management
- Testing and setting permissions and ownership of files
- Systematic deletion of garbage files
- Automated mounting of NFS file systems.
• Other sanity checks.

To use cfengine we need to write a single file which describes the setup of all machines in pool of machines. The interpreter program cfengine need to be compiled on every machine. The program file is distributed to all machines and every one executes the same file. The relevant information is extracted from the file by the interpreter and used to configure each and every machines individually.

2.2 Webmin

One of the modern approaches for administering the Unix/Linux based systems is webmin[3]. Webmin is a web-based interface for system administration for Unix. Using any modern web browser, we can setup user accounts, Apache, DNS, file sharing. Webmin removes the need to manually edit Unix configuration files like /etc/passwd, and lets us manage a system from the console or remotely.

Web based management of host will give lot of flexibility and “ease of use” interfaces to control the servers from remote places. Webmin includes 113 standard modules for administering the remote host such as creating user accounts and shutting down machines editing configuration files including for Apache, DNS, Disk quotas etc. Apart from the basic modules webmin provides the following advanced modules.

• Extended Internet Services: Edit servers handled by Xinetd, a replacement for inetd.

• Filesystem Backup: Backup and restore filesystems using the dump and restore family of commands

• IPsec VPN Configuration : Set up a client or server for an IPsec VPN using FreeSWAN.

• LDAP Server : Manage the OpenLDAP server and objects in its database.

• Linux Firewall : Configure a Linux firewall using iptables. Allows the editing of all tables, chains, rules and options.

• Partitions on Local Disks :Create and edit paritions on local SCSI and IDE disks.

• Sendmail Mail Server: Manage sendmail aliases, masquerading, address rewriting and other features.
2.3 Firewall builder

*Firewall builder*[^1] is the modern GUI based approach to configure and manage the firewall policies both for local and remote machines. This tool supports iptables (netfilter), ipfilter, pf, ipfw, Cisco PIX (FWSM, ASA) and Cisco routers extended access lists. This tool works cross platform and supports many number of ipfilter architectures and policies. Built-in interactive installer uses ssh to communicate with the firewall and can automatically copy generated policy and activate it. It has built-in revision control.

2.4 Puppet

*Puppet*[^2] is a system for automating system administration tasks. *Puppet* is designed to work on most varieties of UNIX-like operating systems. *Puppet* has been developed to help the sysadmin community move to building and sharing mature tools that avoid the duplication of everyone solving the same problem. It does so in two ways:

- It provides a powerful framework to simplify the majority of the technical tasks that sysadmins need to perform.
- The sysadmin work is written as code in *Puppet’s* custom language which is shareable just like any other code.

*Puppet* is usually used in a star formation, with all of the clients talking to one or more central servers. Each client contacts the server periodically (every half hour, by default), gets its latest configuration, and makes sure it is in sync with that configuration. Once done, it can send a report back to the server indicating what, if anything, happened.

Besides the above listed there are many more open source approaches are in development to build a standard architecture for networks management.

[^1]: Firewall builder
[^2]: Puppet
Chapter 3

Problem statement

The motivation for this thesis is the set of real problems faced during the redesign process of network and associated system at Indian Institute of Information Technology and Management kerala, India. The network at includes services such as firewalls, gateways, web, mail and other application services like svn, wiki, and issue tracking. The network was setup nearly 5 years ago, after which so many new services added in an ad-hoc way resulting in error prone operations and malfunctioning of the services.

The common problems during the migration of services and revamping the servers are listed below:

- A major challenge was to design and implement the new network and systems process without affecting routine activities of the institute.
- Traceability for any change was hard to maintain.
- Writing individual scripts and configuration files for every server became a tiresome job resulting in lots of redundancy, confusion, and, errors.
- Change in one system’s configuration was needed to be forwarded across its dependent services.
- Change in any server configuration in the existing network affected all the dependent services.
- Designing test cases for the deployed services was hard.
- Machine information like ip addresses and domain names were hard-coded into the configuration files. So making changes to them were difficult and error prone.
Apart from the above stated problems the redundant data on each machine will make the system administrators confuse, when they have to use separate configuration for each single machine. The real problem of maintenance comes when there are several services running on the same system the then the back up files and modified and updated scripts will become unmanageable.

A deep study on the existing systems revealed the following that factors should be considered.

- Less or no coupling between the administration consoles and the roll back mechanism (which may sometimes entirely were absent).
- Classifying, organizing and maintaining the data to facilitate modifications.
- Lack of work flow leads to difficulty in troubleshooting and traceability.
- Lack of documentation available for the changes made by the administrator.
- Absence of abstraction in the machine configuration files written by hand and hard coded specifications.

Based on the above discussed factors one should need to develop and a theoretical approach to create an architecture which can accommodate all the above functionality. Once the system has a strong and scalable architecture, many applications and protocols can be run on top the system with a minimal or without any changes to the base system. The set of problems faced by us are due to the drawbacks in the base architecture.
Methodology

4.1 Overview

The idea behind the implementation is to organize the whole system information at a central information base. Using the centralized data generate configuration files by a master script. This process of developing the system can be visualized by figure 4.1.

![Figure 4.1: Approach stage by stage](image)

The method for developing the system was organized into 4 following stages:

- Creating a model
- Adding to revision control
- Creating “local build” by build script
- Deployment of the scripts to the respective hosts

The series of steps involved in creating the system comes when we start designing the network. Once the model has been created the modifications done to that data will be under revision control.
4.2 Model

To build this system the administrator first collects and models the key resources elements in the network. The data associated with each machine on a network can be classified like this:

1. Common data shared by all the machines in a network.
2. Data specific to each machine (specific to distribution in BSD based machines).
3. Service specific data in each machine.

According to the above theory the data collected can be classified into different categories and organized. By acquiring the data and classifying the data as above we can create a structure where all the details will be split into different categories (like a Unix file system). This data we call as “base data”. The base data is the data which will tell “which machine does what in a network”. Base data is the data that was organized according to model (see figure. 4.2).

![Diagram of Modeled Systems Data](image)

**Figure 4.2: Modeled Systems Data**

As shown in the figure 4.2 the leaf nodes of each host will contain directories of configuration files specific to that host. And the common node holds configurations split according distribution specific criteria. Along with these files the common node holds the network variables used in the scripts used by each and every host on the network giving the abstraction feature to the whole system. Since the script generated by every build uses the same network definitions we can get rid of hard coding every script. The same network definitions files will be transmitted every host along with generated configurations. These variable are typically “variables in shell”.

11
Each host node on the system also contains a deployment descriptor for generating scripts. The functionality of this file is similar to the *Makefile* in standard *make* utility in Unix style systems. *Make* enables the end user to build and install the package without knowing the details of how that is done - because these details are recorded in the *makefile* that we supply.

### 4.3 Version control

Revision control or version control is the way of management of changes to source code, document, and other programs. This is the most used technique in software development to track the source code changes when the software is being developed. The use of version control system gives flexibility to the user to play around with files, making changes and all the changes will be tracked and can be rolled back at any point of time. One can imagine how chaotic would be the software development without the version control system. Project from multinational companies to big open source projects are completely depends on version control systems. The use of revision control allows the developers work concurrently independent of their physical location.

The typical use of version control system gives the following advantages for development.

- File locking
- Version merging
- Update to previous versions
- Concurrent development.

In our scenario creating a repository is the next step. And importing all the data collected in the same classified order and making local working copies to make changes gives us the above mentioned advantages. Whenever the data changes a relevant commit message will be passed to repository. The message holds the change for tracking.

The commonly used open source implementations of version control systems are:

---

1. *Make* is a tool which controls the generation of executables and other non-source files of a program from the program’s source files. *Make* gets its knowledge of how to build your program from a file called the *makefile*, which lists each of the non-source files and how to compute it from other files. When you write a program, you should write a *makefile* for it, so that it is possible to use *Make* to build and install the program.

2. Some times referred as Source Code Management (SCM) in software engineering.
1. CVS (concurrent versions system)

2. Subversion

3. GNU arch

4.3.1 CVS

Besides its extreme popularity CVS has its own limitations on its performance like changes are tracked per-file instead of per-change, commits are not atomic, renaming files and directories is difficult process, and branching. Some of the maintainers of the original CVS have declared that the CVS code has become too crusty to effectively maintain. Each file connected to CVS is an ordinary file containing some additional information. It is quite natural that the tree of these files repeats the file tree in the local directory. Thus, with CVS you should not be anxious about data loss. In CVS the support of transactions by the principle "all or nothing" is completely absent. For example, when you checkin several files (transfer them to the server), it is possible that the operation will be completed only for some of these files, and will not be completed for the rest (due to conflicts, for example). As a rule, it is sufficient to correct the situation and to repeat the operation for the remaining files (not for all files). That is, the files will be checked in in two steps. No cases of the repository damage due to absence of this functionality were observed.

4.3.2 subversion

These problems led the main CVS developers to start over and create Subversion. GNU arch is a new implementation of version control system and works entirely in a different way compared to CVS and subversion. But Arch has got limitations on file naming and branching conventions. Subversion (SVN) is a new system, intending to be a simple replacement of CVS and has better performance in centralized development compared to other implementations of revision control systems. As a whole, due to some constructive solutions, Subversion really works faster than CVS. It transmits less information through the network and supports more operations for offline mode. We use subversion to achieve the revision control mechanism in this thesis. Subversion-1.6 supports web based view by websvn(a php implementation for viewing the repository contents in browser).

Apart from the basic revision control mechanism some of the advanced features of Subversion are listed below:
1. Works on three major protocols HTTP, SVN, SVN+SSH.

2. Bindings to programming languages. The Subversion APIs come with bindings for many programming languages, such as Python, Perl, Java, and Ruby.

3. Atomic commits support. No part of a commit takes effect until the entire commit has succeeded. Revision numbers are per-commit, not per-file, and commit’s log message is attached to its revision, not stored redundantly in all the files affected by that commit.

4. File locking. Subversion supports locking files so that users can be warned when multiple people try to edit the same file. A file can be marked as requiring a lock before being edited, in which case Subversion will present the file in read-only mode until a lock is acquired.

5. Executable flag is preserved. Subversion notices when a file is executable, and if that file is placed into version control, its executability will be preserved when it it checked out to other locations.

Due to the advantages present in Subversion for this project we adopted Subversion as the revision control mechanism.

4.4 Documentation

Documentation plays a vital role in software development process. A well documented piece of software will be organized better for future change management.

The versioning system with a suitable and relevant commit messages would generate a self documented system. The logs of the versioning system serves as the documentation for changes made. The tools like diff will show the contents changes where as the commit messages reveal the need for changes.

In our project by employing the version control system the following are achieved:

1. The critical part of development, documentation is achieved.

2. The ability to track the changes become powerful.

3. Rolling back to previous states for configuration files without keeping so many files.

4. Data redundancy is by making use of a central repository.
5. Incremental storage nature of version system flushes out the traditional backup system used by administrators.

The whole work done here in this project was documented well and the technical documents were provided along with the this report as the appendices. Please refer the appendix for more on the implementation and design of the system.

4.5 Architecture and Implementation

Once the data repository was setup now the process of generating scripts like a build system needs to done by a master script. The script basically does copying different files, setting up user permissions, setting the owner for each file and creating a local file system structure under temp.

For writing the master script the language can be choice of user interest and command over the language. The language should be able to parse text files and be able to run operating system commands like `chmod`, `chown` with minimal coding. Languages like Perl, scheme, shell have a better approach for interacting with the operating system. However the choice of programming language affects the size and complexity of the code. *Python* is a good choice since all the modern Unix/Linux systems comes with python as built-in interpreter and has powerful set of commands which can invoke underlying operating system functions.

Python is indeed an exciting and powerful language. It has the right combination of performance and features that make writing programs in Python both fun and easy. The following unique features of this language made us to choose python as the scripting language for this thesis study.

1. Simple Easy to Learn

   Python is a simple and minimalistic language. It allows us to concentrate on the solution to the problem rather than the language itself. Python is extremely easy to get started with. Python has an extraordinarily simple syntax.

2. Free and Open Source

   Python is an Free and Open Source Software. In simple terms, we can freely distribute copies of this software, read it’s source code, make changes to it, use pieces of it in new free programs.
3. High-level Language

While writing programs in Python, we never need to bother about the low-level details such as managing the memory used by program.

4. Portable

Due to its open-source nature, Python has been ported to many platforms. All Python programs can work on any of these platforms without requiring any changes at all if we are careful enough to avoid any system-dependent features.

We can run Python on Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and even PocketPC!

5. Interpreted

Python does not need compilation to binary. We can run the program directly from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of our computer and then runs it. All this makes using Python much easier since you don’t have to worry about compiling the program, making sure that the proper libraries are linked and loaded etc. This also makes Python programs much more portable, since we can just copy Python programs onto another computer and it just works!

6. Object Oriented

Python supports procedure-oriented programming as well as object-oriented programming. In procedure-oriented languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In object-oriented languages, the program is built around objects which combine data and functionality. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

7. Embeddable

We can embed Python within C/C++ programs to give 'scripting' capabilities for programs.

8. Extensive Libraries

The Python Standard Library is huge indeed. It will help us in various things involving regular expressions, documentation generation, unit testing, thread-
ing, databases, web browsers, CGI, ftp, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), Tk, and other system-dependent stuff.

Besides, the standard libraries distributed with python, there are various other high-quality libraries such as wxPython, Twisted, Python Imaging Library are available.

Perl programs are easy when they are small and it excels at small hacks and scripts to 'get work done'. However, they quickly become unwieldy once we start writing bigger programs. When compared to Perl, Python programs are definitely simpler, clearer, easier to write and hence more understandable and maintainable. One significant advantage that Perl has, is its huge CPAN library - the Comprehensive Perl Archive Network.

The entire system architecture as shown in 4.3 explains the typical process of script generation and stages in generating scripts from a repository of data. The architecture also shows the impact of software engineering application on computer network maintenance. Hence the principles can be applied effectively to design a robust architecture for networking environments.

4.6 Building the configurations

The whole process of administration divided into 3 phases:

1. Local Build
2. Detection of Host OS
3. Installing on the remote Host

4.6.1 Creating the local build

The build process will be invoked by the script build.py and takes the following arguments as hostname, and configuration directory.

For instance for the host Jupiter the command will be as follows

$ python build.py . Jupiter

The build.py script collects the configuration files from services directory, common data and operating system specific configurations from the common parent directory.
Figure 4.3: Task Management Cycle
Each successful build will create a local directory of scripts for each host in build directory. The build script will read the files and populate the directory to that particular host in the tree structure exactly similar to original file locations for that host.

At one stage of build script generates a file describing the configuration files owner and permissions on the host. Wrong file permissions and owner on configuration files prevents systems service from starting. Since the Unix/Linux system configuration files are much sensitive to the owner and permissions, this file plays an important role when deploying the scripts.

For instance, the resolv.conf file in /etc/ should have the following permissions and owner respectively.

```
-rw-r--r-- 1 root root 114 2009-08-21 11:41 /etc/resolv.conf
```

Once the build has been done the installation by remote installation script which copies all the configuration files from temporary directory to respective hosts by command line. This script takes arguments like hostname, username, host address, and copies to home directory and prompts for a administrative password to copy the files to the respective locations on the file system with proper permissions. The remote deployment of scripts runs on secure shell to remote hosts.

4.7 Checking host system using SNMP and deployment

4.7.1 SNMP check

This section explains how to check remote machine before installation and detecting the target operating system using via SNMP protocol.

Simple Network Management Protocol (SNMP) [6] has evolved as very powerful protocol for monitoring large and medium scale networks. Integrating the present system with SNMP will give extreme exibility to monitor the clients after deployment. SNMP is essentially a request-reply protocol running over UDP (ports 161 and 162), though TCP operation is possible. SNMP is an asymmetric protocol, operating between a management station (smart) and an agent (dumb). The agent is the device being managed - all its software has to do is implement a few simple packet types and a generic get-or-set function on its MIB variables. The management station presents the user interface. Simple management stations can be built with
UNIX command-line utilities. More complex (and expensive) ones collect MIB data over time and use GUIs to draw network maps.

This involves running an executable script written in python which in turn contacts snmp agent on the remote machine. By parsing the output of the snmpwalk on the machine we confirm the installation via remote-install script.

To run the snmp-test command line utility a valid snmp installation should exist on both the invoking machine and target machine. The following script uses the basic command line utilities provided by the net-snmp package, an open source implementation of standard snmp libraries.

command line invoking of the snmp test script:

```
$.snmp-test <remote host>
```

For example once the build has been completed the host named organization.com can be verified like this

```
$.snmp-test organization.com
```

The script handles all possible errors by conditional statements:

The output of script when not enough arguments were supplied to the script is shown in figure 4.4

```
[papa@nanilatex]$.test-machine
Error !! Insufficient arguments
Usage:
./test-machine <remote-host>
Note: remote-host is the Target system to install localbuild
```

**Figure 4.4: Error message: Not enough arguments**

The output of script when the given host name is not valid on the network is show in figure 4.5

```
[papa@nani latex]$.test-machine test
Please wait while contacting the remote system via SNMP.......:

Error !! Please check the host name: test
Details: snmpwalk: Unknown host (test)
```

**Figure 4.5: Error message: Invalid Host**

The output of the script when the script cannot contact the agent SNMP on the target machine is shown in figure 4.6

The script does a snmpwalk on the mentioned host and gives us the information of the remote operating system type and kernel version for all BSD style systems. Additionally it fetches the details of the processor and the interface information.
Figure 4.6: Error message: Timeout

The output of the script when it successfully contacts the remote host is shown in figure 4.7

Figure 4.7: Successful output of snmp-test

After confirming from the script’s output we can go ahead with remote installation by invoking the script at the console.

4.7.2 Deployment of configuration scripts

Once the successful run of the snmp check on the target machine we can now start deploying the scripts using remote installation script.

The remote-install script can be run like the following

$sh remote-install.sh <config-dir> <machine> <remote-host> <username>

For a host named Everest with a user name of johndoe:

$sh remote-install.sh . Everest 192.168.1.200 johndoe

Note: Here the config-dir is “.” means current directory

The above script copies all the scripts to the remote machines along with the file system attributes and ownership details by secure copy and prompts for the user password to deploy them on file system of the remote host.
4.7.3 An example

In this section we explain the generation of scripts for a system used as gateway for a sub-net. Available data for the host are hostname, operating system, domain name servers, routes etc. The typical cycle of operations are shown figure 4.8

![Diagram](image)

Figure 4.8: An example cycle of operations for a host

For a gate way the minimal configuration files needed are:

1. `/etc/networks`
2. `/etc/resolv.conf`
3. `/etc/hostname`
4. `/usr/sbin/local/firewall-preamble-policy`
5. `/usr/sbin/local/route`
6. `/usr/sbin/local/interfaces`
7. `/usr/sbin/local/enable-forwarding`
8. `/usr/sbin/local/nat`

The scripts generated above uses a common network definitions file enables the modularity in writing scripts and ease of handling scripts.
4.8 Backup and Logging

4.8.1 Backup of repository

The backup system for the present administrative data was implemented using the following three tools under Linux system:

1. svn admin utility
2. post commit hooks
3. rsync backup utility
4. Cron scheduler

`svnadmin` command under Unix provides a utility to dump and load the repositories. By taking the dump of a repository and loading it to a new repository will restore all the functionality of the system along with the logs of commits.

To take the dump the of the repository named sys-project can be done by the following command at console

```
$svnadmin dump /path/to/sys-project > dump-file-for-sys-project.dump
```

Loading the dump into a new repository

```
$svnadmin load /path/to/sys-project < dump-file-for-sys-project.dump
```

By using the above listed tools under Linux system we can automate the task for every change in the repository.

For better accuracy and correctness of the dumps, we can take dump after every commit. To implement this we can make use of post commit hooks of the versioning system. And after taking the dump of the system we use `rsync` utility to sync incrementally to a remote backup disk. `rsync` is a fast and efficient file copying tool which uses incremental copying technique. To run the rsync to a remote disk over a periodic interval we use `cron` job scheduler. The cron job scheduler is a simple and powerful scheduler for Linux.

Dumps will be generated after every commit by the post commit hook. The rsync utility syncs the dump files to a backup disk after every periodic time interval by cron scheduler. Typically the cron runs the rsync task after every predefined time period. The whole cycle is shown in Figure 4.9

The post commit hook for generating dump for the repository after every commit. This file can be located in repositories hooks directory with execute permission for the group.
The content of the post-commit file is

```bash
#!/bin/sh
REPOS = "$1"
REV = "$2"
/usr/bin/svnadmin dump "$REPOS" > /home/papa/Public/dumps/dumpfile.dump
```

To setup a cron job in Linux/Unix type the following command at console.

```
$ crontab -e
```

Adding the following job to run to sync the above generated dump file to a remote disk is:

```
/usr/bin/rsync -azh /path/to/dumps/ user@domain.com:/home/user/backup
```

Depending upon the interval for the cron job it will sync the dump to backup disk. When the repository fails the dump can be loaded as shown above.

This backup procedure was taken from svn read-bean book [5]. Please refer the online manual for more configuration details.

### 4.8.2 Logging

The log facility provided by the system here is to track the changes to configuration files by the administrators. The format of the messages is the standard version system log. The log message gives us the description that why the changes are made. The `svn diff` command gives us the standard diff format of the changes.

The log can be seen from the console by the following command:
$ svn log <file name>
For example the following command gives the log messages for all the logs for the modifications of iptables.ipt

$ svn log configuration/machines/redhat/iptables.ipt

The diff command for the subversion to get diff between two different versions of the file is

$ svn diff <file name> -r rev1:rev2
For example

$ svn diff configuration/machines/redhat/iptables.ipt -r 20:34

4.9 Notification to group of administrators

As the whole work of system administration is a collaborative process based approach, every single change made to the repository should notify the group of system administrators. To integrate the revision control system and mail notifications we used the feature called post-commit-hook of the version control system.

Note: The mail notification is not intended to send mails for fail of services as this is not the duty of a management system (assuming all of these tasks will be handled by suitable network monitoring system).

The revision control system we adopted here (subversion) has a feature for running hooks. Example hooks are listed below:

1. post-commit
2. post-lock
3. pre-commit
4. pre-lock
5. pre-unlock
6. start-commit
7. post-revprop-change

The above files reside in hooks directory of the repository and are saved as executable scripts by the svn group and owner. As the name says these hooks will execute our
custom commands or scripts before or after commit. Whenever a commit succeeds
the content of the hook files will be executed. Here in our case we made use of
post-commit hook to send mail notifications.

The notification should be able to send a mail containing the name of the project
in subject along with the changed paths in subject and revision number. The body
of the mail contains the gnu Diff and authors details with time stamp.

The content of post-commit file in our repository contains the following text

```
#!/bin/sh
REPOS = "$1"
REV = "$2"
mailer.py "$REPOS" "$REV" mailer.conf
```

The above command will be executed by Unix/Linux shell after each successful
commit.

Typical directory structure of subversion repository:

```
project/
|-- README.txt
|-- conf/
|-- dav/
|-- db/
|-- format
|-- hooks/
|   -- locks/
```

**Figure 4.10: Subversion Repository Structure**

The software mailer.py and mailer.conf can be downloaded from subversion
mailer [4] project page. The mailer.conf configuration contains settings for the
following:

1. smtp username
2. smtp password
3. smtp server
4. smtp port
5. subject lines

6. log format

7. diff command

8. mail address or list

9. Miscellaneous like groups mappings etc.

mailer.py script is a python script that sends mail to a mail address or a mailing list by parsing the configuration file mailer.conf. The mailer.py script downloaded from subversion distribution will not support the TLS encryption. For this project to support TLS/SSL we need to re write the sender function. Refer Appendix for more details on modified python code snippet.
The organization of the mail:

**Subject**: [project name] < revision number > < changed paths >

**Message body**:

**Author**: admin-campus-1
**Date**: Day Month Date [Time stamp] Year
**New Revision**: Number

Log:

This is sample log message

**Added**:
project/system/current/examples/test.sh
project/system/development/scripts/build.py

**Modified**:
project/system/current/configurations/ssh.conf

The standard GNU diff will be populated here by the mailer.py script when it sends mail.

**Sample mail**:

**Subject**: [svn-sysadmin-group] r20 - in:/thesis-work/current/configuration/build.py
**Author**: gangadhar
**Date**: Fri Oct 2 20:39:45 2009
**New Revision**: 20

Log:

Added file snmp-test for testing the target machine platform. Adding the error handler when there is no deployment descriptor found. Need to be verified when it runs on testbed.
4.10 Testing the installation

The main aim of the thesis work is to create a process based clean approach to maintain the Linux/Unix based configuration files. This approach of administering insists the administrators to follow some of the rules and practices from software engineering to ensure the administration process in an organized manner. To execute and follow the approach involvement of administrators is must in doing several things in administration. Although these kind of administration can be done in many ways, this process based approach ensures a clean and nice, ordered way of administration.

The present system is very flexible to implement in any programming language as it is an approach to do administration with software engineering concepts and pro-
gramming languages. The present system is built with python, Unix shell scripting and some Unix command line utilities, this approach is independent of any language we use for implementation.

The result of following this approach gives the following:

1. Track the changes to configuration files
2. Central repository for holding all the configuration files, in turn gives logging of changes, and roll back mechanism for edits done by administrators.
3. Documentation for all the changes.
4. User manual for using the system.
5. Avoids the redundancy of configuration files on servers.
6. User access control for the data repository allows to write our own access control lists (called acl) for administration.

Though this system gives the scripts in a consistent way they need to be tested once after deployment. Testing for different configurations will be done in different ways for example:

For services runs on the system we use nmap utility in Unix console. Many graphical front ends are developed for nmap such as zenmap and nmapfe.

the nmap scans the ports on the target machine and gives us the information on the remote machine.

As every services like apache daemon, ssh daemon, java rpc services, database services, telnet, will open a port on the server filtered by our rules in configuration files running the nmap on the target host will reveal the opened ports and its filters on a Linux machine.

Example output follows here:

[root@nani ] # nmap -v localhost

Starting Nmap 4.53 ( http://insecure.org ) at 2009-10-19 12:01 IST
Initiating SYN Stealth Scan at 12:01
Scanning papamail.localhost (127.0.0.1) [1714 ports]
Discovered open port 80/tcp on 127.0.0.1
Discovered open port 443/tcp on 127.0.0.1
Discovered open port 8009/tcp on 127.0.0.1
Discovered open port 8080/tcp on 127.0.0.1
Discovered open port 111/tcp on 127.0.0.1
Discovered open port 8443/tcp on 127.0.0.1
Discovered open port 3306/tcp on 127.0.0.1
Completed SYN Stealth Scan at 12:01, 0.10s elapsed (1714 total ports)
Host papamail.localhost (127.0.0.1) appears to be up ... good.
Interesting ports on papamail.localhost (127.0.0.1):
Not shown: 1707 closed ports
PORT STATE SERVICE
80/tcp open http
111/tcp open rpcbind
443/tcp open https
3306/tcp open mysql
8009/tcp open ajp13
8080/tcp open http-proxy
8443/tcp open https-alt

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.149 seconds
Raw packets sent: 1714 (75.416KB) | Rcvd: 3435 (144.284KB)

The nmap output with more options on Netgear wireless router

[root@nani ] nmap -v -sR -P0 -T Aggressive -o nmap.out 10.0.0.1

Starting Nmap 4.53 ( http://insecure.org ) at 2009-10-19 12:34 IST
Initiating ARP Ping Scan at 12:34
Scanning 10.0.0.1 [1 port]
Completed ARP Ping Scan at 12:34, 0.00s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 12:34
Completed Parallel DNS resolution of 1 host. at 12:34, 0.03s elapsed
Initiating SYN Stealth Scan at 12:34
Scanning 10.0.0.1 [1714 ports]
Discovered open port 23/tcp on 10.0.0.1
Discovered open port 80/tcp on 10.0.0.1
Discovered open port 53/tcp on 10.0.0.1
Discovered open port 5000/tcp on 10.0.0.1
Completed SYN Stealth Scan at 12:34, 1.75s elapsed (1714 total ports)
Initiating RPCGrind Scan against 10.0.0.1 at 12:34
Completed RPCGrind Scan against 10.0.0.1 at 12:34, 0.90s elapsed (4 ports)
Host 10.0.0.1 appears to be up ... good.
Interesting ports on 10.0.0.1:
Not shown: 1710 closed ports
PORT STATE SERVICE VERSION
23/tcp open telnet
53/tcp open domain
80/tcp open http
5000/tcp open UPnP
MAC Address: 00:1E:2A:6B:57:46 (Netgear)

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 2.872 seconds
Raw packets sent: 1719 (75.634KB) | Rcvd: 1715 (68.618KB)

For the routing scripts we use traceroute utility for routes defined by the our routing scripts.
1: 10.0.0.2 (10.0.0.2) 0.183ms pmtu 1500
1: 10.0.0.1 (10.0.0.1) 1.218ms
1: 10.0.0.1 (10.0.0.1) 3.629ms
2: 192.168.1.1 (192.168.1.1) 3.259ms
3: 192.168.1.1 (192.168.1.1) 3.411ms pmtu 1464
3: ABTS-KK-Dynamic-001.0.167.122.airtelbroadband.in (122.167.0.1) 44.270ms asymm 4
4: ABTS-KK-Static-045.32.166.122.airtelbroadband.in (122.166.32.45) 43.254ms
5: ABTS-KK-Static-009.32.166.122.airtelbroadband.in (122.166.32.9) 44.663ms
6: 122.175.255.29 (122.175.255.29) 44.891ms
7: 59.145.0.50 (59.145.0.50) 71.675ms asymm 8
8: 203.101.100.233 (203.101.100.233) 72.194ms asymm 9
9: 125.17.224.134 (125.17.224.134) 86.729ms asymm 10
10: 202.141.131.234 (202.141.131.234) 77.258ms asymm 11

For testing the firewalls there are graphical front ends for the tools as well mentioned above. Network vulnerability scanners like cheops, nessus will be nice in detecting intrusions from inside and outside as well.
4.10.1 Testing the scripts before installation

A firewall is only as good as its implementation. In today’s dynamic world of Internet access, it is easy to make mistakes during the implementation process. By auditing the firewall setup, we can ensure that the firewall is enforcing what we expect it to, in a secure manner. The best way of ensuring the accuracy of the scripts we generate is to audit the configurations with a formal rule base.

Auditing the firewalls is a two-stage process:

1. Auditing and inspecting the firewall itself

2. Writing a rule base for the expected output and compare both.

Access control lists represent an important first line of defense on most networks, since they are commonly used on routers to limit the protocols allowed to pass to host systems behind the router. Firewalk is an open source tool that helps in verifying routers ACLs are actually doing what you intended them to do. It can also be used as part of our security tool set for penetration testing and providing documented verification of ACLs, as well as rule sets on firewalls.

One of the best tools for auditing our firewall and firewall rule base is a good port scanner. By scanning the ports as shown in above output samples we can reassure the performance and expectations with our installation. The most critical parts of the network includes DMZ’s. The above two things need to be verified by the administrators carefully before deploying the firewalls. Auditing the best practice in developing a good firewall script. The above will apply for the other services as well in network infrastructures.

The automation of the audit of the firewalls and other configuration scripts from the generated scripts itself a big task we feel it is out of the scope of this project. But as this topic completes the whole system approach in a Software engineering life cycle much more work can be done on the above said.
4.11 Systems extensibility

Apart from the design discussed here the present system can be extended to great extent by adding support to other protocols and automating the script generation. One of the powerful system configuration tool cfengine can be used in conjunction with the above system to automate the overall process. As the versioning system is part of the whole system the admin group should be notified after each successful commit done to it. This can be achieved by integrating a post commit hook to the versioning system. As many of the software systems uses data bases for data reliability and integrity definitely a data base in our case will bring a powerful architectural base. Instead of splitting the data on a file system data, it can be inserted in to a database and can be queried by the script to populate the configuration files. Cron, The most popular and widely used job scheduler in Unix based systems. Cron enables unix users to execute commands or scripts (groups of commands) automatically at a specified time/date. The minimal use of cron can download emails and connect the host to Internet etc.

4.11.1 Future work with snmp

Due to time limitation the project implementation has so far been detecting the remote host operating system. By introducing the SNMP which is a protocol based, operating which implements SNMP(most of modern the OS have SNMP support Built-in) can be addressed via SNMP. The operating system of the device being managed can be queried using snmp get requests. By detecting the operating system type the above discussed system can deploy scripts specific to that operating system. The above said system will now contains different kinds of scripts for different kind of operating systems. Basically the system will have very high chances of accommodating large varieties of operating systems, thus the scalability will be vertically and horizontally increased.

4.11.2 Database back end

Using an appropriate data base will give so many advantages compared to that of a flat file system of current implementation. The most efficient and reliable data security and integrity can be achieved by using a data base for large data storage. Our systems data can be split in to a data base at the back end and can be queried by the deployment system so that all the data will be persistent and less effective for file system changes.
The data base data can be dumped and backed up more efficiently. The data can be inserted into different tables and keys by designing proper integrity checks, which will save coding effort in master script.

4.11.3 User Interface

At present the system provides a command line interface and for version control the subversion commands at the Unix command line interface.
Summary and conclusions

The results of this thesis demonstrate the crucial role of configurations files of hosts in network and gives an approach to organize them in a systematic way. And there is much more needs to be done and this thesis gives the design possibility to one such system. As stated above the scope of the project can be increased dynamically with the number of hosts. As the system does data organization in such a way that modification and change of the data can be tracked easily, scalability of the system to many and different types of systems can achieved.

**Note:** The system is not a *host configuration system or a central management station for managing hosts in a network*. This system gives a solution for managing the configuration files of hosts in better fashion by adopting some of the software engineering principles. Integrating this system with host configuration system or management stations yields a management station. The management stations can take over the work after transferring the configuration scripts to each hosts which leads to totally automated system.

The following are achieved by adopting this approach in computer networks (assuming the version control system is always present).

1. Data redundancy avoided by use central configuration repository.
2. Version control system gives roll back facility.
3. Documentation available for every commit to repository easier to track changes.
4. The system supports all variants of Gnu/Linux based system.
5. Reliable and systematic approach achieved to maintain configuration files.
The following needs to be worked out and remains as the future work for this thesis:

1. Platform Independent application of the present system can be achieved by employing protocols like SNMP (In progress).

2. Data base model implementation for systems data storage (future work) after detailed study the pros and cons of various databases.

3. Automatic testing of generated scripts.
Configuration files

Example shell variables used as network definitions

nw=192.168.0.0 # nw is the name of a network
nw-mask=255.255.255.0 # mask
sep-ww=192.168.0.1 # gateway
nw-www=192.168.0.2 # web server
nwmail=192.168.0.3 # mail server

An example firewall script which uses the variables defined in network definitions system wide file

iptables -A FORWARD -d $source -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -d $source -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -d $source -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -s $source2 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -s $source2 -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -s $source2 -p tcp --dport 53 -j ACCEPT
iptables -A FORWARD -s $source2 -p udp --dport 53 -j ACCEPT
Example pre-routing and post-routing commands after using global definitions

```
iptables -A PREROUTING -t nat -d $external-ip -p tcp --dport 80 -j DNAT --to $internal-ip1
iptables -A PREROUTING -t nat -d $external-ip -p tcp --dport 22 -j DNAT --to $internal-ip2
iptables -A PREROUTING -t nat -d $external-ip -p tcp --dport 22 -j DNAT --to $internal-ip3
iptables -A POSTROUTING -t nat -o $external -j SNAT --to $internal
```

Example routing and interface definition commands in a gateway:

```
/sbin/ifconfig $interface-1 down
/sbin/ifconfig $interface-2 down
/sbin/ifconfig $interface-1 netmask $nw-mask $main-nw
/sbin/ifconfig $interface-2 netmask $sub-mask $sub-nw
/sbin/ifconfig $interface-3 netmask $mask-app1 $app-1
route add -net $nw-one gw $nw-two
route add default gw $isp-gw
```
User Manual

B.1 How to build and export machine configuration

This document describes the process that enables the automation of the generation and export of machine-specific configurations to the machine.

B.1.1 Who is allowed to build the script?

You will need to have svn access to the snag system repository in order to check out the configuration. If you’re reading this file, you already probably have that privilege.

B.1.2 Who is allowed to export?

You will need to have an certificate accessible ssh user account on the server to which you wish to export the script.

B.1.3 How to generate the build directory

The shape of the configuration directory is as follows:

To run the build script it’s best to be in the config directory. From here you run the build script as follows:

./build.py . MACHINE
The first argument, . here, is the the root of the config directory. The second argument is the name of the machine.

The script makes the following assumptions:

1. You have /usr/bin/python installed.
2. MACHINE is a directory under config/machines/
3. config/machines/MACHINE/files is a file that contains the copy descriptors. (See description below)

B.1.4 Format of config/machines/MACHINE/files

Each line is either a directory descriptor or a file copy descriptor. Here are the example descriptors for a host.

# For directories:

```
etc 0755 root root
usr 0755 root root
usr/local 0755 root root
usr/local/sbin 0755 root root
```

# For files:

```
nw/interfaces 0644 root root usr/local/sbin
nw/netroute 0755 root root usr/local/sbin
```
When the script successfully runs, it creates the build directory and populates it as follows:

![Build directory diagram]

**Figure B.2: Build directory**

### B.1.5 attributes.sh

This file is a shell script generated by build.py. The script is a sequence of chown and chmod commands. When invoked on the remote machine, this script sets the owner, group and file permissions of files before they are copied to the final destination on the remote machine (usually /).

### B.1.6 SNMP host check

The script called "snmp-host-check" will check the host operating system before deploying the built scripts.

Example output of above script after successful run:

```
---------------------------------------------------------------
Remote Operating system Linux nani 2.6.27.25-78.2.56.fc9.i686
---------------------------------------------------------------
Other details follows
```

```
GenuineIntel: Genuine Intel(R) CPU T2080 @ 1.73GHz
GenuineIntel: Genuine Intel(R) CPU T2080 @ 1.73GHz
```

42
network interface lo
network interface wlan0
network interface eth0
SCSI disk (/dev/sda)

Guessing that there’s a floating point co-processor

To continue installation of the local build to the remote please invoke remote-install.sh

B.1.7 Exporting configurations to remote machine

change to the configuration directory. Then run the following at shell prompt:

./remote-install.sh . MACHINE REMOTEHOST USERNAME

This will install the config files under build/MACHINE/tmp to REMOTEHOST:/
The script will invoke a script using sudo on REMOTEHOST:, so type your user
password of REMOTEHOST when prompted.

Run remote-install.sh with extreme care!

If you’re careless, you can easily trash your REMOTEHOST. Make sure you under-
stand and agree with the files under MACHINE/tmp and the names, directories,
permissions and owner and group attributes of the files in MACHINE/attributes.sh.
Subversion mail notifier

C.0.8 mailer.py modified source code

In this project we used the google’s mailing system to deliver mails to a mailing list. Since google uses TLS encryption to send mails we modified the code to support the TLS encryption. After downloading the mailer.py file from subversion website find the function `smtplib.SMTP` and we edited the code as following. SMTP mail servers of the organizations can be configured to send mails without passwords and encryption. This modification is not necessary for all mail servers.

```python
server = smtplib.SMTP(self.cfg.general.smtp_hostname,
                      self.cfg.general.smtp_port)
if self.cfg.is_set('general.smtp_username'):
    server.ehlo()
if self.cfg.general.smtp_use_tls:
    server.starttls()
    server.ehlo()
server.login(self.cfg.general.smtp_username,
             self.cfg.general.smtp_password)
server.sendmail(self.from_addr, self.to_addrs, self.buffer.getvalue())
server.rset()
server.quit()
```

C.0.9 Configuration settings for mail

The mailer.conf file contains the following options: The sections in blue color defines configuration setting for mailer.py
[general]

smtp_hostname = smtp.organization.com
smtp_username = svnnotifier@gmail.com
smtp_password = xxxxxxxxxx
smtp_port = 587
smtp_use_tls = 1

[defaults]

diff = /usr/bin/diff -u -L %(label_from)s -L %(label_to)s %(from)s %(to)s
commit_subject_prefix = [svn-project] # this comes as part of subject line

propchange_subject_prefix =
lock_subject_prefix =
unlock_subject_prefix =
from_addr = nobody@gmail.com
to_addr = sysadmins@organization.com
generate_diffs = add copy modify
show_nonmatching_paths = yes

[maps]

from_addr = [authors]
to_addr = [mailing-lists]

[authors]

john = john@example.com
doe = doe@example.com

[mailing-lists]

m1 = version-commits@example.com
m2 = admins-commits@example.com
Bibliography


[9] W.M. Fitzgerald, S.N. Foley, and M O’Foghlu. Network access control interop-
eration using semantic web techniques. *6th International Workshop on Secu-
ritiy in Information Systems, (WOSIS 2008), Barcelona, Spain, June, 2008.*
2009.

ceedings of the First International Workshop on Secure Semantic Web, 2008,


Managing network devices like host systems. *Network Operations and Man-
October 19, 2009.

http://www.ee.kth.se/1cn/anm08/posters/5.pdf, Accessed on: October 19,
2009.