The following paper was originally published
in the Proceedings of the
Tenth USENIX System Administration Conference (LISA X)
Chicago, IL, USA, Sept. 29 - Oct. 4, 1996

Automating the Administration of Heterogeneous LANs

Michael Fisk
New Mexico Institute of Mining and Technology

Automating the Administration of Heterogeneous LANs

Michael Fisk – New Mexico Institute of Mining and Technology

ABSTRACT

The areas of machine configuration and software package installation and maintenance have been frequent areas of work in recent years. This paper describes a hybrid system developed to address both problems and more. The resulting system is designed to reduce the complexity of the administration of a large network of computers down to that of the administration of a few heterogeneous systems.

In particular, this system allows machines to be maintained without ever having to manually change files on their disks. Systems can also be upgraded, installed from scratch, or recovered with a minimum of effort. The system described is designed to be extremely general and applicable to virtually all versions of UNIX and UNIX-like operating systems.

Design Goals

The development of this system was driven by two primary goals. First, consistency should be maintained between machines whenever possible. Machines should have the same configurations and same software unless specifically designed otherwise. At a configuration level, this means assuring that similar machines use the same nameservers, run the same daemons, and generally operate identically. At the software level, it means that the same version of a package should be installed the same way on all architectures and all machines whenever possible.

Second, the process of installing and upgrading machines should be automated as much as possible. New hardware, operating systems, and configuration changes should be easily assimilated into the configuration of the local network. Upgrading to a new operating system should not require scheduling a large block of down time and a correspondingly large amount of human labor. Similarly, it should be easy to install any single new machine so that it matches all similar machines on the network.

More specifically, the resulting system should simplify the following tasks:

- Installing and configuring new machines without forgetting any of the myriad of small details that make a machine operate as expected with a network.
- Making small configuration changes to all current machines.
- Performing major OS upgrades to all current machines.
- Upgrading services such as mail or NNTP.
- Migrating services between servers. As we upgrade our servers and separate services onto discreet "toasters," moving a service, such as mail or NNTP, to a new machine was a painstaking process involving too much trial and error. Making sure everything was configured correctly on the new server and removing all superfluous things from the old server was too labor intensive.
- Maintaining a server-based `/usr/local` filesystem for multiple architectures. As with most sites, our clients mount a `/usr/local` filesystem from a server. Due to the extremely diverse needs of our user community, our `/usr/local` is quite large and exists for four incompatible flavors of UNIX on different hardware platforms. Installing a package for all architectures so that it behaved identically on each was too time consuming.
- Migrating packages from a server-based `/usr/local` filesystem to local disk on machines with larger disks. filesystem to local disk on machines with larger disks.
- Maintaining specialized packages on individual machines. There was a growing need to be able to install and maintain machines in peoples' offices with special software that is not available over the network to all users.

In addition, we sensed the opportunity to use the same infrastructure to provide the following functions that had also been identified as tasks that were overly troublesome or redundant.

- Maintain legal DNS files. These files seemed to be constantly incorrect and often contained illegal syntax from some of our less experience system and network administrators.
- Maintain correct `/etc/ethers` file. This file is routinely neglected, but used for some network management functionality.

Previous Work

OMNICONF [Imazu] isolates configuration files from operating system files, and provides mechanisms for storing and then restoring the configuration files. It does not, however, aid in installing the operating system itself. It is also not designed for changes to be made to a central template and automatically appear on each workstation. Instead, changes are supposed to be made on a sample workstation and the differences stored for application to other machines. *OMNICONF* also seems to lack an easy way of sharing some, but not all, of a configuration between similar, but not identical machines.

GeNUAdmin [Harlander] provides extensive mechanisms for maintaining configuration files on machines, but is not designed to maintain whole installations of large software packages. *Config* [Rouillard1, Eirich] is better suited to maintaining both configuration files and whole software packages, but depends on a potentially complex `rdist` file and a primitive database for describing machines so that they may be grouped for `rdist`. In addition, the use of `rdist` implies a push rather than a pull mechanism. This can cause problems when changes are pushed while machines are down.

Sasify [Shaddock] uses a set of action scripts to perform arbitrary operations on machines. The appropriate action scripts are run using hostclasses representing logical groupings of machines. *Sasify* also uses a more attractive pull paradigm at reboot to insure that changes get propagated. Unfortunately, *Sasify* lacks the infrastructure needed for managing large sets of software packages for multiple architectures.

lcfg [Anderson] is designed to solve many of the same problems that this system addresses, and therefore provides much of the same functionality. It is unclear, however, how well *lcfg* handles the requirement for different software to be installed on different machines. *lcfg* makes extensive use of dynamic configuration at boot-time. It is therefore intentionally less sophisticated in maintaining static configuration information.

What is desired is a system that uses *Depot*-style [Rouillard2] mechanisms for storing and arranging software, but that also provides ways of maintaining machine-specific configuration files and software subsets. As discussed below, the system should also require clients to periodically pull new changes from a server rather than depending on a server being able to push them.

New Concepts

Static vs. Dynamic

As discussed by Anderson [Anderson], there is a trade-off between statically configuring machines by putting their configurations on local disk and dynamically configuring machines by having them query a server at boot time.

Out site has experimented with both types of operations. Our experience has led us to believe that for reliability and performance, all configuration information should be stored locally on the machine. This allows machines to boot without the presence of the server. While our main server is generally up, we have been working hard to make our clients as independent as possible. Not only does this reduce load on the server, but in cases where large numbers of machines reboot after power outages or similar disasters, it allows machines to boot correctly even if the server is not up yet.

Changes to configuration information should be made on a central server and then propagated to the client. This is most robustly done by pushing information to all reachable clients **and** by having all clients check the servers periodically or at boot-time. This guarantees that machines that are down or unreachable won't miss changes, but also allows for the immediate propagation of changes without intensive, frequent queries of the server.

Representation of Configurations

Many previous systems for machine configurations are based on performing a series of sequential operations on a machine [Shaddock, Eirich]. All operations must be saved and applied to machines to insure proper configuration. This leads to the problem of determining whether certain actions have been superseded by others. It was the conclusion of the author that the configuration of any typical UNIX workstation can be represented solely on the contents of its filesystems. Therefore, the configuration process need not contain a series of sequential operations that need to be run on the machine. Instead, a machine's configuration can be completely represented as the union of the correct set of files.

Therefore, the system is built around a set of separate packages. Each package is the files necessary for some logically discreet functionality. For instance, X11R6, Emacs, and INN are all typical packages. In order to preserve generality in the system, the vanilla operating system, as installed from original media, is stored as a package. When a machine is installed from scratch, the operating system package is installed first. Site-specific configuration packages are then loaded on top of that.

Previous work [Rouillard1, Eirich] has been done based on enumerating files and directories that should be copied or installed on machines or groups of machines. Packages offer a more implicit way of doing this since these files are often stored together for convenience anyway. Using packages, directory layout and organization can be represented in their most natural states, directories.

A central database is maintained to define which packages should be installed on any given machine. Hostclass [Fletcher] definitions are awkward for adding new configurations. It is harder to locate all classes that a given machine belongs to and to duplicate those for a new machine. In addition, classes are then usually associated with actions or groups of files in a separate database. Because of this, it is difficult to evaluate the effect of adding a machine to a given class.

For these reasons, the database is designed to make use of inheritance hierarchies. A machine is defined as an object. It can contain arbitrary variable/value pairs as well as a list of other objects to inherit variable/value pairs from. Objects can be defined for any logical group of machines that share some configuration information. This structure allows a machine to have a large amount of information associated with it, while still allowing short, but flexible, definitions for machines. It also makes cloning the configuration of a new machine off of an existing one easy. Additionally, exceptions to standard configurations are allowed for.

Creating Packages

The operating system package is generally made by installing a machine from scratch from the vendor's CD. Once the operating system is installed and configured to the point where it can talk to the network, the filesystem is dumped into a package directory on the installation server.

Our site runs Linux on over 100 PC's. A few of these still have very small hard disks. For this reason, we have two different versions of the Slackware package. One version includes virtually all packages in Slackware. The other is a stripped-down version that is sufficient to boot a machine, but will fit on a small disk.

Our site also has a couple of DEC Alpha servers. Since DEC releases an operating system update as often as we install a machine, we have not placed the OS in a package. Instead, we install the machine from CD as is usually done and then just execute a partial installation to configure the machine. Since we will typically not install this same operating system version again, we do not bother copying the OS into a package on the installation server as we normally would.

Structure

The system is composed of three main components, described below.

Machine Database

The machine database, or *Machdb* as it is called, defines all machine-specific variables. These range from IP address to nameserver to a list of packages to install. The database is currently implemented as a text file defining any number of objects. Each object can contain any number of variable/value pairs and can inherit from any number of other objects, recursively. This technique allows a definition for a machine to consist of only a few unique values such as IP and MAC address. All other values can be inherited from other objects.

The database allows set operations on variables in order to support combinations of different software sets as well as exceptions to standard configuration. For example, assume that all client machines are defined to inherit from an object by the name of `standard-client` that defines

```
PACKAGES=amd ypclient X11R6 bash emacs
```

Now assume that TeX is not a commonly used package at the site, but that it is required on some machines. The object for a client may inherit from `standard-client` to get the default list and also define `PACKAGES+=tex`. The += operator will append rather than destructively setting the value for that variable. In addition, that client can also define `PACKAGES-=ypclient` to prevent it from getting the YP configuration files even though `ypclient` is listed in the `PACKAGES` definition in `standard-client`.

The database is queried via the *machdb* program which is a Perl script that parses the file, expands inheritances, and produces responses in various formats. One such format prints out all hosts in DNS record format suitable for direct use by BIND. There is also an option to check all machine objects for the existence of certain required fields. A set of sample database entries and query results are included as an appendix.

Packagelink

Packagelink is a Perl program that takes hierarchically organized package directories for multiple architectures and builds a single target directory for users to access. It was originally designed to be run on a fileserver to build a `/usr/local` filesystem containing hard links to all appropriate files for that architecture. In the last two years, it has been expanded to build mirror filesystems by building the target filesystem on a different machine than the packages reside on and copying files instead of linking them. *Packagelink* is designed to be extremely flexible and can build not only `/usr/local`, but any filesystem. This method of operation is used to build `/` on a machine from a set of packages on the installation server.

Packagelink is designed to operate on running systems. For that reason, it is careful to not delete or change existing files until it is sure that the current version is different from the version to be installed. Therefore, it can be run on a well-configured system and will make no unnecessary changes that would hinder other processes.

While copying files, if *Packagelink* encounters any files ending in a particular suffix (`.plprocess` by default), it will run those files through an internal preprocessor. Any strings of the form '`##VARIABLE##`' will be replaced with the corresponding value specified by a '`-DVARIABLE="VALUE"`' command line option. Conveniently, *machdb* can output a list of `-D` arguments for all variables defined for a machine. This preprocessing serves as a substitute for having a script generate machine-specific configuration files or for needing a separate package for every machine.

The ## syntax is also used since # is the traditional comment character for UNIX configuration files. Thus, a single comprehensive `inetd.conf` (or other configuration file) can be created for an entire group of machines. Services that will only run on some machines can be put in lines beginning with a `##SERVICENAME##` directive. This will remain in the file unless '`-DSERVICENAME=""`' is specified as an argument to *Packagelink*. If so, the comment will be removed and the service activated.

Most services, however, can be optionally installed using a System V style directory of start scripts where each service has its own script. Script names are alphanumerically ordered so that they are started in the correct order. Each script is stored in the package that it starts. Therefore, if a package is not installed, the script will not be placed on the machine and will not be run. BSD-style systems can have this functionality added by appending a simple *for* loop to the end of `rc.local`. This eliminates the need to maintain different `rc.local` scripts for each machine.

Packagelink has used various algorithms for determining how to combine packages into the proper filesystem. Currently it scans all packages and builds a database in memory of every file in every package. This database is used to resolve conflicts between different packages. Different packages can be given different priorities so that if a file exists in more than one package, the version in the package of highest priority will be installed. If duplicate directories exist, *Packagelink* can be configured to always merge them or to only merge packages of the same priority.

The database built by scanning packages is dumped into a file for subsequent executions of *Packagelink*. Using this saved database, future duplicates can be resolved without scanning all previously installed packages.

A *Packageunlink* program is also planned. This program will remove all files for a specified package and replace them with any duplicate files that had previously been overridden because they were in packages of lower priority.

A *Packagecheck* program will also be written to verify that all files on a machine came from a package and are correct. This will provide some of the functionality of *Tripwire* [Kim].

Installation Program

The main program, affectionately called *gutinteg*, is also written in Perl. In its simpler mode of operation, the default, the program configures an already running machine by performing a partial install. This is usually done by running the script by hand or from a periodic cron job. The program probes a machine for its MAC addresses using `ifconfig` and `dmesg`. The *Machdb* entry for that MAC address is then loaded. *Packagelink* is then called to install software on the machine. Using `/` as the target filesystem, *Packagelink* copies all files onto the machine. *Gutinteg* then reinstalls the boot block or boot loader on the root partition.

For complete installs, the machine is booted from an alternate device. External SCSI drives, boot floppies, and network filesystems have all been used. The install scripts is added to the end of the normal boot sequence. In addition to the normal steps of a partial install, in this mode, gutinteg makes use of the `PARTITION` variable in the machine database to partition and format all disks. A sample value for a Solaris machine is:

```
c0t3d0: (a 500 ufs /) (b 500 swap /tmp)
c0t1d0: (h 1000 ufs /home data)
```

This value specifies a 400 cylinder partition for `/`, and a 500 cylinder partition for swap (to be mounted as `/tmp` on machines that support Sun's tmpfs [Snyder]). A 1000 cylinder partition for `/home` is created on a second disk. The '`data`' flag means that this partition will not be formatted during installation.

The partition letters are abstract. On operating systems with numbered partitions, they will automatically be converted to 0-6 or 1-7 where appropriate. On operating systems that traditionally have a 3rd partition containing the entire disk, one will be automatically created. On Linux machines, extended partitions and other DOS-isms are also handled automatically.

After partitioning and formatting disks, *Gutinteg* installs the appropriate operating system package(s), and proceeds with the steps of a partial installation. Finally, operating system dependent functions such as configuring `/dev` and `/devices` are performed and an `fstab` or equivalent file is produced from the `PARTITION` list.

DOS

Our group is also responsible for maintaining a Novell network of DOS machines. We decided to require that Linux be installed on all of these PCs so that we can use UNIX tools to maintain the local DOS partitions. The machines automatically reboot every night to Linux, where the DOS partition is mounted as `/dosc`. Currently, a cron job runs just after that to fix any changes that users may have made to the local disk during the day. This functionality is being replaced by the new system which will maintain `/dosc` as it maintains every other directory.

Usage Experience

A previous version of the system was used to upgrade 26 Sparc workstations to SunOS 4.1.4 in the fall of 1995. We also used this opportunity to have the system repartition all of the disks at the same time. A single, external, boot disk was used to boot each machine for installation. The process was so trivial, that our User Consultants (who are not trained in UNIX system administration) upgraded all of the workstations without assistance from any of the system administrators. While the process could have been accelerated by remotely booting several of the machines off the network at once, this method allowed us to keep all but one of the workstations up at all times for use.

The current version has been used for all new Linux workstations purchased or evaluated in the last few months and all Linux and Sun machines on our test network. A complete reinstall of over 100 Linux machines was performed this summer by one programmer during a weekend.

Complete installations can take anywhere from 30 min to over an hour depending on the size of the system, network traffic, etc. The main bottleneck for reinstalling a large number of systems is the ability to do so simultaneously. Ramdisks on Linux machines and network root partitions that can be used by multiple machines simultaneously have been key to allowing us this capability.

In addition, we are currently working to populate the installation server with packages for all of the key services on our main servers. Once this has been completed, the system will be used to reinstall all of our servers.

Once all of our machines have been reinstalled using the system, we will abandon all other methods of maintaining local machines, and switch to running the partial installation script automatically on a regular basis.

Conclusions

The system described in this paper has allowed our system administration staff to stop maintaining individual machines and instead maintain a single large template of packages and configuration files. Since the files are still stored in traditional directory hierarchies, the learning-curve for maintaining the new system is minimal. As a result, maintaining a large network of machines is becoming less related to the number of machines, and instead proportional only to the number of different architectures, operating systems, and software packages.

Availability

The three tools comprising this system, *gutinteg*, *machdb*, and *packagelink* are all freely available Perl programs. They are, however, not yet designed for public consumption. Anxious readers are welcome to contact the author for information on how to retrieve current versions. Note that they will largely be unsupported software, but I am always willing to work with other users and, if necessary, modify the programs to make them useful to a wider audience.

Acknowledgments

The author would like to thank K. Scott Rowe for his continuing work with and experience using this system, Tony Heaton for his assistance in testing it, and James Robnett for allowing the author to pursue good ideas when he found them.

Author Information

Michael Fisk developed this system while employed by the New Mexico Institute of Mining and Technology's Computer Center. He has since completed his B.S. in Computer Science from that university and is currently a staff member at Los Alamos National Laboratory in the Network Engineering Group of the Computing, Information, and Communications Division. His current work includes introducing the topic of this paper to system administrators at the Lab, prototyping protected open networks, and researching other areas of internetworking infrastructure. He can be reached by e-mail at *mfisk@lanl.gov* or by postal mail to: Michael Fisk; CIC-5/MS B255; Los Alamos National Laboratory; Los Alamos, NM 87545.

References

[Anderson] Anderson, Paul, "Towards a High-Level Machine Configuration System", *LISA VIII Proceedings*, 1994.

[Eirich] Eirich, Thomas, "Beam: A Tool for Flexible Software Update", *LISA VIII Proceedings*, 1994.

[Fletcher] Fletcher, Mark, "doit: A Network Software Management Tool", *LISA VI Proceedings*, 1992.

[Harlander] Harlander, Dr. Magnus, "Central System Administration in a Heterogeneous Unix Environment: GeNUAdmin", *LISA VIII Proceedings*, 1994.

[Imazu] Imazu Hideyo, "OMNICONF – Making OS Upgrades and Disk Crash Recovery Easier", *LISA VIII Proceedings*, 1994.

[Kim] Kim, Gene H. and Spafford, Eugene H., "Experiences with tripwire: Using integrity checkers for intrusion detection", *SANS III Proceedings*, 1994.

[Rouillard1] Rouillard, John P. and Martin, Richard B., "Config: A Mechanism for Installing and Tracking System Configurations", *LISA VIII Proceedings*, 1994.

[Rouillard2] Rouillard, John P. and Martin, Richard B., "Depot-Lite: A Mechanism for Managing Software", *LISA VIII Proceedings*, 1994.

[Shaddock] Shaddock, Michael E. and Mitchell, Michael C. and Harrison, Helen E., "How to Upgrade 1500 Workstations on Saturday, and Still Have Time to Mow the Yard on Sunday", *LISA IX Proceedings*, 1995.

[Snyder] Snyder, Peter *tmpfs: A Virtual Memory File System*, http://www.sun.ca/white-papers/tmpfs.html, Sun Microsystems, Inc.

Appendix: Sample Database Definitions

```
underdog: dot3 Standard486 speare20
        IP=129.138.3.162
        MAC=00:40:33:2d:08:3f

dot3:   classC tcct.nmt.edu
        GATEWAY=129.138.3.1
        BROADCAST=129.138.3.255
        NETWORK=129.138.3.0

classC:
        NETMASK=255.255.255.0

tcct.nmt.edu:
        NAMESERVER=129.138.3.220
        NAMESERVER2=129.138.4.216
        DOMAIN=tcct.nmt.edu
        YPSERVER=teal

speare20:
        LOCATION=speare20
        PRINTER=speare16

Standard486:    i486 mach32 at1500

at1500:
        NIC=ne1500t
        PACKAGES+=at1500

i486:   full-linux
        CPU=i486
        ARCH=i486

mach32:
        VIDEOCARD=mach32
        PACKAGES+=mach32

full-linux:     small-linux
        OSREV=Slakware3.0
        PACKAGES+=xfig transfig

small-linux:    base-linux
        OSREV=small-Slakware3.0
        PACKAGES+=nntpclient

base-linux:     dos
        OS=linux
        OSREV=Slakware3.0
        ROOTFS=hda2

dos:
        PACKAGES+=dos
```

Sample Database Query Result

```
q-bert.IP="129.138.3.162"
q-bert.MAC="00:40:33:2d:08:3f"
q-bert.PARTITION="hda: (a 51 msdos) (b 421 ext2) (c 53 swap)"
q-bert.PACKAGES="xfig transfig nntpclient at1500 mach32"
q-bert.LOCATION="speare20"
q-bert.PRINTER="speare16"
q-bert.OSREV="Slakware3.0"
q-bert.NIC="ne1500t"
q-bert.VIDEOCARD="mach32"
q-bert.CPU="i486"
q-bert.ARCH="i486"

q-bert.GATEWAY="129.138.3.1"
q-bert.BROADCAST="129.138.3.255"
q-bert.NETWORK="129.138.3.0"
q-bert.NAMESERVER="129.138.3.220"
q-bert.NAMESERVER2="129.138.4.216"
q-bert.DOMAIN="tcct.nmt.edu"
q-bert.YPSERVER="teal"
q-bert.NETMASK="255.255.255.0"
q-bert.OS="linux"
q-bert.ROOTFS="hda2"
```