Cloning Customized Hosts (or Customizing Cloned Hosts)

George M. Jones & Steven M. Romig – The Ohio State University

ABSTRACT

In an environment consisting largely of many similar machines, it is certainly advantageous to treat these machines as clones of a master copy (the so-called "cookie cutter" approach). One of the problems with this approach is that vendor supplied tools typically do not allow the system administrator to easily change the master copy or to customize individual hosts after they have been cloned. In this paper we will discuss a general methodology for handling clones that includes provisions for variations among the machines. We will also discuss `update-client` and `build-server`, two tools that exemplify this methodology in the OSU-CIS environment. Finally we will analyze the advantages and disadvantages of these tools and describe our future plans.

Introduction

We have a fairly large number of nearly identical hosts at The Ohio State University Computer and Information Sciences Department (OSU-CIS). We have 260 diskless Sun SLCs (our Sun clients), several SparcStation 1s (used as clients), 22 Sun file servers (20 3/180s, 1 4/280, 1 4/330), 25 SparcStation 2 file servers that are slowly replacing the Sun 3/180s and the 4/280, and about 2 dozen other UNIX hosts. The Sun clients are nearly identical in their hardware configuration. Most of the clients are SLCs with 8 megabytes of memory, and most are diskless. Most of the 3/180 servers have identical SMD disk configurations, the rest have nearly identical SCSI disk setups. All of the SparcStation 2 servers are identical. We have been using the "cookie cutter" approach since we installed our first Suns (three Sun 2s) long, long ago.

Initially we used Sun's standard tools, which are now called `suninstall` and `add_client`, to setup servers and their diskless clients. As our environment grew more complicated we discovered that the tools that Sun provided were insufficient for maintaining our environment for several reasons.

Both `suninstall` and `add_client` were designed to install a SunOS distribution on a wide variety of platforms at all Sun sites. The tools are very general and require a certain amount of expert knowledge to get the configuration of a given server or client correct. This is fine for a tool that is used once in a while by a trained administrator to install a distribution for the first time. However, it is difficult for a relatively untrained operator to use these tools to correctly setup a server or client (as they often have to do to fix things). Ideally one would like `suninstall` and `add_client` to automatically use predefined configuration files to reduce the number of critical choices that untrained workers have to make in setting up servers and clients.

It is also difficult to change the "master copy" (or prototype) that `suninstall` and `add_client` use to create servers and clients. In the case of `suninstall` the prototype is the SunOS set of distribution tapes (or worse, the unwriteable CD!), and it would be unwieldy at best to make changes to `/bin` or whatnot and rewrite the tapes. `add_client` uses a prototype root directory (stored on the server) to create new clients. For most of the changes that we make to the clients, it would be quite easy to simply change the on-line prototype and be done with it. However, some of the changes would require changing the `add_client` script itself – most notably, we use a file called `/etc/rc.config` to contain host and network information (addresses, default gateways, broadcast address, netmask). It would be nice to be able to create a new prototype that included local changes to the standard distribution.

Further, `suninstall` and `add_client` make no provision for individual variation. Although we have 260 SLC clients that are essentially alike in most respects, 66 of them have significant changes that would have to be reinstalled by hand after running `add_client`. Some have printers and require special printcap files, some have swapping disks and require special kernels, some have special daemons that run on them, some have different `/etc/ttytab` files and so on. Similarly, our servers are alike in most respects, and yet different in some subtle but important ways (special daemons, printers and so on). We could handle this in part by creating shell scripts to make those specific changes to the standard client or server layout, and then run the shell scripts as post-processing after using the Sun tools.

In summary, `suninstall` and `add_client` are insufficient for the task of building many cloned servers and clients because they do not allow you to make your own prototype, they do

not provide for automatic customization of the hosts being installed, and because one cannot easily select a predefined configuration and apply it to a host.

A General Approach To The Problem

We would like to clarify that we are not trying to solve the problem of setting up a server or client for the first time. `suninstall` and `add_client` do this work admirably - they are fairly easy to use, and they are flexible enough to handle almost any sort of configuration.

The problem we are trying to solve is this: having installed the software and made our local changes, we want to be able to create a prototype out of the system we have set up, and use that prototype to setup subsequent servers (or clients), or to repair damaged servers and clients.

There are two parts to our proposed solution. We need a tool to create a prototype based on an existing server or client which is to serve as a model for subsequent clients and servers. We also need a tool to build a server or client based on that prototype. The building tool should allow one to easily select the configuration to use, and it should have a provision to use host-specific customizations that will allow a site administrator to record and retain individual variation in a collection of hosts based on a common prototype.

At OSU-CIS, we have created tools that conform to this model to solve this problem for diskful servers and diskless clients. The rest of this paper describes the tools for building servers and clients (respectively) and then summarizes our experiences with these tools and our future plans for them.

The Build-Server Script

Build-Server Overview

The `build-server` script is used to load a standard set of software onto Sun fileservers from a standard software prototype. We form the prototype by using `suninstall` to install the SunOS distribution on a server. We add our local modifications and test the system until it passes muster. Then we save a copy of the `/` and `/export` file systems with GNU `tar`. These `tar` files constitute the prototype from which other servers are created (or fixed).

To build a new server, the machine that is to have software loaded onto its disks is booted as a diskless client (though it would be possible, in principal, to boot it off a portable disk and use that to do the update). Once the server is running as a client, the `build-server` script is run to load software onto the disks.

The script knows how to do disk formatting, surface analysis, disk partitioning, create filesystems, check filesystem consistency, load software onto the partitions, select and install default kernels and how to install boot blocks. Based on user input, it can do as little as load software back onto one partition (for example, if a partition was badly damaged and had to be newfs'd), or as much as analyzing, formatting, partitioning, and loading software onto all disks on the system (for example, when we setup a new server).

Making a Master for Build-Server

Making a prototype for the `build-server` script consists entirely of saving copies of the "important" file systems (such as `/` and `/export`) with GNU `tar`. These copies should be made from a server that is up to date and which has been thoroughly tested, since any bugs present on the server will be saved in the copies and inflicted on servers created from them.

Build-Server Configuration

The `build-server` script uses a configuration file that allows the user to establish a mapping between a server's hostname and a particular server configuration. Among the elements of a configuration that may be specified are the architecture type, the number and type of network interfaces, the number and type of disks, the partitioning to use, and the path to a nonstandard kernel to use.

All servers are created equal, but they do not stay that way very long. Although the servers are built from the same prototype files, there are various changes that the `build-server` script makes to create their identity and to install special differences (like services that run on one server but not another).

The first changes that must be made to all servers are the host name and network address. We have created a configuration file, `/etc/rc.config`, that defines shell variables for items that vary from server to server, such as the hostname, network address, names of network interface(s), etc. This is used by all of the other RC files, and allows us to gather most of the host specific information into a single file where it can be easily edited. One of the first things that the script does after loading the root filesystem is to modify the values of these shell variables (`ed(1)` is your friend!) to reflect the proper values for the particular server in question.

We maintain identical `/etc/rc` and `/etc/rc.local` files on all servers. Boot time actions that are particular to certain servers are invoked from `/etc/rc.local`, keyed off the host name.

Benefits, History and Experience with Build-Server

The `build-server` script collects expert knowledge of system configurations and the processes necessary to perform system setup in a form that both documents them and makes it possible for nonexperts to perform the processes. Some of the tasks that are automated are tedious and error prone if performed manually. Other tools such as `suninstall` have similar benefits but do not make it easy for a non-expert to choose and apply various configurations to particular hosts.

The original version of `build-servers` was written when we were faced with the task of converting 20+ servers from supporting 250 Sun 3/50s running SunOS 3.5 to 250 SparcStation SLCs running SunOS 4.0. All of the servers were converted in a two week period by two people, with as many as four updates running in parallel (one has to be mindful of resource contention limited items such as tape drives, and of network limitations if one is running more than a single copy of `build-server` simultaneously!). The script is currently being updated to load software onto the SparcStation 2s with SCSI disks. The script itself required almost no modification. It was sufficient to simply write new configuration file entries.

Problems With Build-Server and Future Plans

`build-servers` attempts to be a user-friendly, easy to use interface for doing most initial system configuration tasks (after the hardware is set up), but reformatting a disk is still a big deal, running 10 passes of disk analysis still takes a long time, and it will always be possible to screw things up in a big way if such tools are not used with care by people who have a certain level of competence and training.

`build-servers` does not replace `suninstall` – it is not nearly so versatile. Its sole purpose is to apply stored configuration information and a changeable software prototype to repair or install diskful servers with optional host specific customizations. It also does not handle the needs of subsequent updates once the servers are running – that is what our update-servers script is for (see below).

Currently, the process of setting up a server to boot as a client in our environment is not trivial. It requires a good deal of knowledge of things such as tftpd, rarpd, Ethernet addresses etc. It is not currently feasible to tell one of our operators "go boot server X off of server Y." Also, adding new servers to the network (not just rebuilding old ones) is somewhat involved. There are a large number of changes that must be made to the nameserver databases, `/.rhosts`, `/etc/hosts.equiv`, `/etc/fstab`, NIS (nee YP) databases, etc. Something almost always gets missed on the first pass. This process could stand some automation. We are planning to set up the SparcStation 2s so that they can be easily be booted as diskless clients from other servers, from which you could run the `build-server` script to fix any problems.

We intend to add support for scripts specific to a particular server to be run after the standard software is loaded onto the server. This will allow for further customization of individual servers while retaining the benefits of having a common starting point and being able to reload a server at any time.

The script currently only loads system software and locally installed software packages, not user files. We intend to add the ability to have it load user files from our regular backup tapes.

We may move away from using tape for the master copy and keep a copy on disk updated at regular intervals.

In the longer term we may consider adding some "intelligence" to the script so that it can automatically check the sanity of things like the contents of the root filesystem and reload the software if needed. If a file system goes up in smoke, it still takes a certain amount of expertise to determine whether one should reload the file system from scratch to fix it, or just restore a few missing files from the latest backups.

The Update-Client Script

Update-Client Overview

The `update-client` script is used for setting up diskless sun workstations. It uses a client root prototype (stored as a GNU `tar` archive of what should be in a client's root directory), a client configuration database (the `/etc/client-info` file) and optional scripts that are used to customize specific clients.

To build a new client, one need only setup the prototype (with the `make-master` script), edit the client configuration file, and run `update-client clientname`. The `update-client` script will remove the client root directory (if it exists), create a fresh root from the master, setup the various architecture and host specific features according to the `/etc/client-info` file, and execute the client customization script (if it exists). The process is simple and foolproof and results in a client root that is set up the way that it "is supposed to be". Our operations staff frequently uses this command to recreate broken client partitions.

Making a Master For Update-Client

A script called `make-master` is used to create a prototype GNU `tar` copy of the root directory of a client that is to serve as the prototype for the others. The `make-master` script copies a named client's root directory, sanitizes it by

removing log files and architecture specific things (`vmunix`, `sbin` and so on), and uses GNU `tar` to save a copy. The script will refuse to make a master if the client that is named has a customization script (see below), since one would not want to propagate those customizations through the master copy to the rest of the clients. The saved prototype file is stored in a common location on all servers, where `update-client` can find it. The same prototype file can be used for Sun clients of any architecture, since the architecture specific changes are simple and are easily handled by the `update-client` script itself.

Update-Client Configuration and Customization

`update-client` gets its host information from the `/etc/client-info` file. The client-info file describes common characteristics of each of the clients, such as hostname, the location of its root directory and swap file on its server, which server it boots from, and its binary and kernel architectures. Here is an excerpt from the client-info file on a typical server:

```
KEYWORDS server root swap arch karch\
  fstab proto swapsize time broadcast

DEFAULT
  arch     sun4
  karch    sun4c
  fstab    /usr/local/config/fstab
  proto    /usr/local/config/master.gtar
  swapsize 30m

#
# Fish entries
#
DEFAULT
  server    fish
  time      0
  broadcast 128.146.29.255

carp.cis.ohio-state.edu
  root      /export/clients0/carp
  swap      /export/clients0/carp.swap
bass.cis.ohio-state.edu
  root      /export/clients0/bass
  swap      /export/clients0/bass.swap
```

This example shows most of the features of the client-info file. The `keywords` entry names the fields that are allowed to appear in the file. This is used for some simple error checking. The `default` entry sets default values for fields. In this case, we set the default binary and kernel architectures, the location of the `fstab` file and the prototype root, and the size of the swapping file. In a second `default` entry, we set the server, time (used when making crontab entries that run on all clients), and the broadcast address. The `default` entries can be used repeatedly to set new defaults throughout the file. Finally we have the client entries. Each entry starts at the beginning of a line in the file, and contains of all the name/value pairs that appear until the next entry starts. This file essentially lists all of the information needed to create a client root and swap for it to boot.

In addition to the normal client information recorded in the client-info file, a client can also have special customization information stored in a shell script in the `/usr/local/config/specific` directory. When `update-client` finishes the root directory for a client, it looks for a shell script in the specific directory with the same name as the client. If it exists, it is executed. The shell script is run with its current working directory set to the client's root directory, and can do anything that it wants to that client's files. For example, Steve has a client customization script for his workstation that rewrites the `/etc/fstab` file to remove most of the NFS mounts, rewrites `/etc/rc` to start `amd` (an automounter), and installs a kernel that supports the SCSI disks, tapes and CDROM that are attached to his SLC. The client config scripts are saved between client updates, so if you want to make permanent changes to a client, you simply create a customization script that does the right things, and subsequent client updates will retain your changes on top of the new client setup.

Benefits, History and Experience with Update-Client

`Update-client` is intended to be easy to use, and since the current version only takes one argument (the name of the client to update) it is hard to get it wrong. The operations staff at OSU-CIS uses it frequently to fix broken clients. It also provides for a fairly simple means for various people on the staff to set up customized workstations – we have customization scripts for 66 clients at this time. It also allows us to easily and thoroughly make changes to the root directories of all of the clients – it takes about 45 minutes to update the 260+ client root directories.

Problems with Update-Client and Future Plans

Although `update-client` does work well, it still has its problems. The main problem is that there are still some bits of configuration information in the script itself. For example, there is a fragment of the base `fstab` file used on clients, and parts of the base `crontab` file. We have to edit the script itself to change these and a few other things. It is currently being rewritten to move yet more of the configuration information out of the script itself.

`Update-client` is not a replacement for Sun's `add_client` script. It is not versatile enough to use for the task of creating new client setups, especially on new releases of the OS.

Miscellania

When a newly created server is booted for the first time a script called `update-all-clients` is run automatically from `/etc/rc` to set up any diskless clients that that server has. `update-all-clients` simply gets a list of the names of this server's clients and runs `update-client` on each to create its root directory and swapping file.

Once we have created a bunch of nearly identical servers, we also need a mechanism to keep the software consistent. We use a shell script called `update-servers` to propagate changes from a "master" server (typically the one used to create the software prototype used by `build-server`) to the other servers. The `update-servers` script is a "smart" wrapper for `rdist(1)`. It accepts directory names on its command line and checks to see that it is being run from the correct host, checks for unsafe directories or files (such as `/etc` – it would be bad if the master's `rc.config` file were pushed to the other servers!), creates a `rdist` script to update those directories, and invokes `rdist` to update the servers.

Concluding Remarks

These scripts have proven to be incredibly useful. They help us maintain an environment where the various hosts are constructed from a consistent base (the clone concept) while allowing us the freedom to introduce individual variations. They make it possible for relatively untrained staff members to build new servers and clients. The ability to make new prototypes is also very useful, but the provision for preserving and applying customizations has probably been the biggest advantage.

We are continuing to improve both of these tools by making them more general and by moving most of the configuration information out of the scripts and into configuration files. We do not know whether these specific scripts would be suitable as they are for use at other sites or not. It should be easy to modify them to work in other environments.

In the long run, it might be better to add configuration selection and customization to `suninstall` and `add_client`, which would eliminate the need for extra tools. One could then use `add_client` in the usual way to create an initial client, change it to reflect local requirements, and make a master prototype from that client. Then one could use `add_client` with different options to apply that prototype and any desired customizations to other clients to build or fix them. Similar things could be done with `suninstall` so that one could either answer a bunch of questions and make a new server configuration, or apply an existing prototype and customizations to a server to repair or build it.

Acknowledgments

Steve Romig wrote the original specifications for both `build-server` and `update-client`. Tom Fine wrote the original `build-server` script. George Jones worked on it with him, and Tom is working on the newest version. Steve wrote `update-client` and has been slowly and continually improving it. Various members of the OSU-CIS staff have made numerous suggestions, requests and helpful comments about both scripts.

Author Information

George Jones is a member of the software support staff at the CIS Department at The Ohio State University. His email address is george@cis.ohio-state.edu, his U.S. Mail address is The Ohio State University; Department of Computer and Information Science; 2036 Neil Avenue Mall; Columbus, OH 43210, and his telephone number is 614-292-7325.

Steve Romig is the software staff manager for the CIS Department at The Ohio State University. His main professional interests are in simplifying and automating system administration tasks and in computer security. His electronic mail address is romig@cis.ohio-state.edu, his U.S. Mail address is The Ohio State University; Department of Computer and Information Science; 2036 Neil Avenue Mall; Columbus, OH 43210, and his telephone number is 614-292-8018.

References

Sun Microsystems, SunOS Reference Manual Volume i, section 1: "rdist(1)", July 17, 1986.

Sun Microsystems, SunOS Reference Manual Volume iii, section 8: "add_client(8)", January 13, 1990.

Sun Microsystems, SunOS Reference Manual Volume iii, section 8: "suninstall(8)", January 13, 1990.

Sun Microsystems, SunOS 4.1.1 Release & Install, "Installing the SunOS", 1990