Automating UNIX and Linux Administration

KIRK BAUER

Apress™

ISBN (pbk): 1-59059-212-3

Printed and bound in the United States of America 10987654321

Technical Reviewers: Nate Campi, Erik Melander, Alf Wachsmann

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Copy Editor: Rebecca Rider

Production Manager: Kari Brooks

Proofreader: Laura Cheu

Compositor: Susan Glinert Stevens

Indexer: Kevin Broccoli

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 6

Automatic System Configuration

YOU HAVE JUST INSTALLED and configured a fresh copy of an operating system on a new system and you are now ready to automate its configuration. A UNIX system's configuration is usually contained in hundreds, if not thousands, of individual files. Many of these files contain system-level configurations such as network settings, device information, drive mappings, and so on. The rest of the configuration belongs to various applications you have installed on the system, such as web servers and user applications.

A thousand configuration files might seem a bit overwhelming at first. Luckily, most of these files will never need to be modified because the defaults are suitable for most situations. One significant advantage to automating the administration of UNIX systems is that most of the configuration files are text and not binary. In some cases, it doesn't matter because you will use a utility program to create the file for you. In other cases, however, you will have to directly modify the file, and then you will be glad when the file is plain text.

This chapter first explores a custom solution that can be adapted exactly to your needs. The custom solution may be perfect for small groups of systems, stand-alone "network appliances," and situations with very specific or unique requirements. I expand on the concept of system grouping throughout the custom solution examples. Also, I supply many examples of files that should be configured on most systems. For these reasons, even readers who have decided to use cfengine may want to skim through this chapter's custom examples.

The latter half of this chapter provides a comprehensive introduction to the GNU cfengine. This system automation/configuration application is very powerful, flexible, and useful—particularly for larger networks of differing systems. cfengine uses the pull method that allows each system to configure itself independently of other systems. This is more reliable and scalable in most situations than the push method used in the custom example. Regardless of your thoughts on cfengine right now, you should read through the introduction starting in section 6.4, "Configuring Systems with GNU cfengine," which fully describes all of the benefits it can provide.

6.1 What Do I Configure?

The configuration on a system can be divided into several categories:

Customized files: Files whose contents differ from system to system.

Dynamic files: Files that are not the system default and may change several times over the life of a system.

Default files: Files whose default contents are adequate for your purposes.

Static files: Files that need to be customized only once.

Filesystem components: Symbolic links, directories, device files, file permissions, and so on (filesystem components).

6.1.1 Customized Files

There are certain files that are going to be different on each machine. The most obvious are the network settings, particularly the system's IP address. Other settings that might be different include the firewall configuration, the services that should be run at startup, and so on.

These files are the most important part of the configuration automation system. Even on a relatively small number of systems, any files that are different from system to system can be major sources of difficulties. Attempting to manually create those files is bound to result in errors. If you want to make a change to such a file, you have to make the change on each system manually. Or, in some cases, you may copy the new file onto each system and make the customizations manually on each machine.

Since the file on each system is, by definition, different, it is nearly impossible to verify that any given file is correct, apart from painstakingly examining each file manually. For these reasons, customized files are the focus of this chapter.

6.1.2 Dynamic Files

These files are the same on each machine, yet they are modified from time to time. For these files, you could simply maintain a master system and then copy any new versions of the file(s) out to the other systems using a program such as Secure Copy (SCP). It is, however, easy to forget to perform the copy, or neglect a particular system in the copy process. It is also easy to make changes to one of the systems and forget to make those changes to the master system.

You can usually distribute these dynamic files using the methods within this chapter. Alternatively, the methods for data synchronization discussed in Chapter 7 can be used for this type of file. The size and number of files determine which approach is best in your situation. The methods in this chapter are, in general, suited for smaller numbers of smaller files *or* files that don't change too often. You should usually accomplish sharing or distributing applications and their data with the tools in Chapter 7.

6.1.3 Default Files

In an ideal world, our automation system would handle every file on any system. We would be able to verify that any given file is, indeed, correct. If, in the future, we decided to modify a new file, it would already be part of the automation system and could just be modified on the configuration server.

In the real world, however, it is not usually worth your effort to worry about files you don't expect to change. If you change your mind in the future, you can always expand your automation system. If you still want to verify that the other files have not been changed from their original values, you can use the tools discussed in Chapter 11.

6.1.4 Static Files

You need to modify some files' content from their default, but after you do, this content will remain static from that point forward. You can handle these files in a variety of ways:

- You can configured the file during the system's initial setup as discussed in Chapter 5. `root`'s static `~/.ssh/authorized_keys` file is a good candidate for this method.
- You can handle the file the same as other data as discussed in Chapter 7.
- You can distribute the file using the techniques discussed in this chapter.

You may want to consider using the most appropriate of these methods for each individual file. Alternatively, you can choose one method for all files for simplicity and consistency.

6.1.5 Filesystem Components

Although most system configuration takes the form of configuration files, the actual filesystem itself can also be important. The`~/.ssh/authorized_keys` file is a configuration file, for example, but before it can be created, the `~/.ssh` directory must already exist. In addition, the permissions on the directory and the file must be correct, otherwise the SSH daemon ignores the file.

Another example of making configuration changes directly on the file system involves operating systems using System V init scripts. System V init scripts use symbolic links to determine which services start at bootup. These symbolic links are also part of the system's configuration, but they don't really make up a configuration file.

I also discuss the creation, management, and verification of these filesystem-level configuration items in this chapter.

6.2 Configuration Principles to Follow

The principles you will follow when you build your configuration automation system are the same as those you will continue to see throughout this book:

- The system should operate on as many different operating systems as necessary.
- The system should enable you to place systems into one or more classes and treat each class differently.
- The system should allow you to update the configuration on each system whenever and as often as you like. If nothing needs to be done, nothing should be changed.
- The system should be able to automatically update the various machines and/or update machines on demand, and possibly even update individual machines on demand.
- The system needs to be self-documenting. At the very least, you should be able to see what you changed two years ago. Hopefully, you will also know why.
- The system should be self-repairing. If a file on a system is deleted or the permissions are changed, the system should repair that problem.
- If you are automating a site that has many system administrators or has a high turnover rate, then having a single tool for almost everything is very important.

Normally, I would include a verification requirement in that list. For example, it would be nice to have a command that could verify that every system is properly configured. In this case, however, we have specified that the system should be able to be run on demand and not cause any harm. So, in lieu of verifying the system's configuration, you could just activate it and, after it runs, you will know everything is configured just as it should be.

In some situations, however, you still may want to have real verify abilities. You might want to see if something has changed, possibly back up that change, and then update the configuration. If this is required, you could always back up the original version of every configuration file before you update it. You could then use the `diff` command (or equivalent) to determine if the files differ (and to get the actual differences). This facility could be added to the methods described in this chapter with a little additional effort.

6.3 Creating a Custom Configuration Approach

In this section, we develop a system that configures a variety of machines based on a master set of configuration files and scripts. This configuration information, along with the files, scripts, and templates that compose the system, resides on a master server. This server is assumed to have passwordless Secure Shell (SSH) access to each system it needs to configure.

I am a big fan of directory structures. By using a complex directory structure, I actually make the contents of each file much simpler. The script that must process the data also tends to be simpler. It also makes the whole system more modular and easier to manage with source-control systems like CVS.

On the negative side, directory structures can be intimidating at first. Instead of one configuration file, you have to look through various directories, which is particularly difficult if you don't have a guide to get you started. It also can be a pain when you have to modify several different files to make a small change (which sometimes does happen).

All in all, I feel that directory structures are worth it. I like the structure and order. I like having simple text files that can be edited and managed by shell scripts. Also, I often find that I only edit a small subset of files subset of files—most files are generally left alone.

For these reasons, the custom approach in this chapter uses a directory structure. As with any program, there are many ways this structure could be written, each with its own advantages and disadvantages. The following example should provide you with a good starting point and/or good ideas for a script of your own.

6.3.1 Defining the Files and Directory Structure

For this example, all files will be under the `/usr/local/etc/` directory. The following directory layout will be used for this example and its associated scripts:

```
/usr/local/etc/
|-- all_hosts
|-- group_list
|-- hosts/
|   |-- desktop
|   |-- linux
|   `-- web
`-- sysconfig/
    |-- conf/
    |   |-- all
    |   |-- groups/
    |   |   |-- desktop
    |   |   |-- linux
    |   |   `-- web
    |   `-- systems/
    |       |-- host1
    |       |-- host2
    |       `-- host3
    |-- files/
    |   |-- all/
    |   |-- desktop/
    |   |-- linux/
    |   `-- web/
    |-- scripts/
    |   |-- all/
    |   |-- desktop/
    |   |-- linux/
    |   `-- web/
    `-- templates/
        |-- all/
        |-- desktop/
        |-- linux/
        `-- web/
```

The `all_hosts` file contains a list of all hosts (by name) in your network. The `group_list` file contains a list of all possible groups (each system can be a member of one or more of these groups—i.e., one system might run Linux, be used as a desktop, and run a web server, so it would be a member of all three example groups).

The hosts/ directory contains one file per group that contains the list of hostnames of the systems in each group. All of these files were created and some were used in previous chapters.

The sample groups used for this example are desktop, linux, and web. These example groups were also discussed and used in Chapter 4.

Everything under the sysconfig/ directory is new for this chapter. I recommend using CVS to manage this directory. If you make a bad modification and corrupt a template, you can always use CVS to see what you changed and/or revert back to the working template. CVS will be introduced in section 7.9.

6.3.1.1 The conf/ Directory

Each file within this directory and its subdirectories contains any number of configuration settings. Each file is in a format that can be sourced directly by bash scripts. It can also be easily parsed by any other language. Here is a sample system-specific file:

```
IP_ADDRESS=192.168.10.11
NETMASK="255.255.0.0"
HOSTNAME='host1'
```

As you can see, quotes can be omitted if there are no spaces or special characters (such as ;, !, &, and others). Double quotes escape these characters but still allow for variable expansion. Single quotes disable any processing of the argument completely.

The conf/all file contains settings that are common for all hosts. Like any setting, they may be static or they may change from time to time.

The conf/groups/ directory contains settings specific to a given group. Any settings in that file override any settings in the all file for hosts in that group. In most cases, this directory also contains settings that are not found in the all file because they do not apply to all hosts. Each group does not need a file, so groups that do not have any special settings need not be represented here.

If settings within different group files conflict, the situation is a bit more tricky. Obviously there is only a problem if a system is in two groups that have different values for a specific setting. It is best if this doesn't happen. If it does, the groups are processed alphabetically so that the setting in the last alphabetical group is used.

The conf/systems/ directory contains one file per host. The filename needs to be the system's hostname. Not all systems need to be present—only the systems with specific settings. Any values in a system-specific file override any values in any other file (the global or group-specific files).

6.3.1.2 The files/ Directory

The `files/` directory contains several subdirectories. One of them contains files that belong on all hosts (the `all/` directory). Also one directory per group contains files that belong on hosts within that group.

Each file should be the exact file you want on the proper hosts. The file is placed on each host unmodified except for certain metadata that is removed from the file. This metadata specifies the destination location and permissions for the file. It needs to be in a format that is not likely to be found in any normal file. In this case, I use the sequence `__:VAR=VALUE` that must appear first in a line. Here is an example `/etc/resolv.conf` file:

```
__:LOCATION=/etc/resolv.conf
__:PERMS=0755
__:USERGROUP=root.root
search mydomain.com
nameserver 192.168.1.2
```

This example file should be named `resolv.conf` and be placed in the `sysconfig/all/` directory. This causes this file to be placed on every host as `/etc/resolv.conf` and its permissions to be set to `0755`. The first three lines are not present in the destination file (nor are any lines of this format) because this metadata does not belong in the final configuration files.

6.3.1.3 The scripts/ Directory

Just like the `files/` directory, this directory has an `all/` directory and one directory for each group. Each directory can have any number of scripts (or any executable program) that are executed on the appropriate systems based on their group membership. Both the groups and each set of scripts in each group are processed alphabetically.

6.3.1.4 The templates/ Directory

The `templates/` directory structure is identical to the `files/` directory. The only functional difference is that each file in the `templates/` directory is processed for possible substitution of the various settings from the configuration files. That is what separates templates from regular files—templates can contain any number of tokens that are replaced with real values during the configuration process.

The substitutions are actually made by processing the file using the bash shell. This allows us to have a wide variety of powerful substitution operators (everything that the bash shell supports). Here is an example template:

```
IP_ADDRESS="${IP_ADDRESS}"
SUBNET="${SUBNET}"
```

This template is very simple and contains just some host-specific network configuration parameters. The values for these particular settings would be in a host-specific file in `conf/systems/`. Here is an example line in a template file that uses one of the more powerful substitutions the bash shell supports:

```
SUBNET="${SUBNET:-255.255.0.0}"
```

The `${var:-default}` substitution string uses the value in the `SUBNET` configuration item. If that value is not set, `255.255.0.0` is used as the default value. With this method, only a system with a unique netmask has its netmask set in its system-specific configuration file. Any other systems have the default value substituted in place of this substitution string.

Of course, you can also put a default value in the `conf/all` file, which operates in the same way. You have the option of using whichever method you think is the best for values such as this that are usually the same across multiple systems but can be different in some cases.

Here is another possible line from a template:

```
IP_ADDRESS="$(sed -n 's/ $HOST$//p' /usr/local/var/prepare_system/allocated)"
```

This line uses the `$(cmd)` substitution string, which causes the specified command to be executed and the output to be substituted in its place. In this example, the template assumes the `HOST` variable is set to the system's hostname (and it will be set by the main script that will be processing the template). The file containing the list of IP addresses and the host to which they are assigned (created by our scripts in Chapter 4) are processed and the host's IP address is isolated using the `sed` command. Remember that in this custom configuration system, the template is processed on the central configuration server, not on the destination system.

One disadvantage of this particular template system is that certain reserved environment variables cannot be used to store settings. You should not, for example, have a variable named `PATH` or `IFS`. The bash man page has a complete listing of the special environment variables used by the shell. As long as you avoid using these as variable names, you will not have any problems.

The greatest disadvantage of this type of template system is that you cannot usually push any shell scripts out using this system. A shell script is likely to have substitutions that will be processed when the template is parsed, and as a result,

the script will be corrupted. This limitation can be overcome by pushing the shell script out as a file (and not as a template). The script could then read any configuration from a separate configuration file. This is, arguably, a better way to do it anyway.

You could also use a completely different method for the template files. I often use a Perl script to process templates and use something like `__VARIABLE__` for substitution values. Note that this method requires more coding and does not allow nested substitutions (although it could if I chose unique opening and closing delimiters such as `__{VARIABLE}__`).

6.3.2 Defining the Configuration Logic

We will have one configuration script (`config_all_systems`) that runs on a central server and performs various actions on the remote systems based on the information in the directory structure discussed in the previous section.

The logic for this script is lengthy but straightforward. The list of hosts will be read from `/usr/local/etc/all_hosts` and the following actions will need to be performed for each host:

1. Determine which groups the host belongs to by processing the files in the `hosts/` directory.

2. Load the configuration from the `conf/all` file.

3. Load the configuration from each necessary group file in the `conf/groups/` directory.

4. If it exists, load the system-specific configuration from the appropriate file in the `conf/systems/` directory.

5. Transfer any files in the `files/all/` directory to the remote system and into the location specified in each file.

6. For each group the system belongs to, transfer any files in the `files/groupname/` directory to the remote system and into the location specified in each file.

7. Process any templates in the `templates/all/` directory locally and then transfer them to the remote system and into the location specified in each file.

8. For each group the system belongs to, process any templates in the `templates/groupname/` directory locally and then transfer them to the remote system and into the location specified in each file.

9. Transfer any scripts in the `scripts/all/` directory to the remote system and then execute them on the remote system.

10. For each group the system belongs to, transfer any scripts in the `scripts/groupname/` directory to the remote system and then execute them on the remote system.

6.3.3 Presenting the Configuration Script

This system is divided into two scripts: one script, called `config_all_systems`, executes the `configure_system` script for each system and summarizes the results. The `configure_system` script, does the actual configuration on any one system.

One good reason for this separation is simplicity (each smaller script is simpler than one big script would be). A better reason for separate scripts is that they isolate environment variables. Each host needs to start with a clean slate for its variables. One host might, for example, have a special `SUBNET` value, while the next host wants that variable unset so that the default variable will be used (when using the `${var:-default}` substitution string). By executing each host in a separate shell, you will not have any problems with the settings for one host being left over when you configure other hosts.

I present the `config_all_systems` script first because it is fairly simple. Next in this section I cover the `configure_system` script and explain all of the components within that complicated script.

6.3.3.1 The config_all_systems Script

The `config_all_systems` script presented in this section simply executes the second script for each host. It finishes by summarizing the results, displaying any failed hosts by name, and exiting with `0` only if all hosts were successful.

```
#!/bin/bash

# This file should have one host per line
host_list="/usr/local/etc/all_hosts"
```

```
# This is the script that configures each host
configure_system="/usr/local/sbin/configure_system.sh"

# Process each host
success_count=0
failure_count=0
failure_list=''
for host in `cat $host_list` ; do
   echo
   if $configure_system "$host" ; then
      echo " Host $host Configuration Successful!"
      success_count="$[$success_count+1]"
   else
      echo " Host $host Configuration FAILED!"
      failure_count="$[$failure_count+1]"
      failure_list="$failure_list $host"
   fi
done

# Display summary
echo
echo "Total of $[$success_count+$failure_count] systems processed:"
echo " $success_count Success(es)"
echo " $failure_count Failure(s)"
echo
[ -n "$failure_list" ] && {
   echo "$failure_count host(s) failed:"
   for host in $failure_list ; do
      echo " $host"
   done
   exit 1
}
exit 0
```

NOTE You can find the code samples for this chapter In the Downloads section of the Apress web site (`http://www.apress.com`).

There shouldn't be much in this script that you have not seen before. One exception might be the use of bash's mathematical capability. The `$[expr]` expression evaluates `expr` as a mathematical equation. The results are substituted in its place. This is used to count the successes and failures to be reported at the end of the script.

6.3.3.2 The configure_system Functions

Although there are many distinct steps for each host, many of them are related and can be implemented using functions to reduce the amount of code in the script. As any developer knows, functions are a very good thing. Using them means that you have less code to write, debug, and maintain.

The following functions should be placed into a file that can be included (sourced) by the `configure_system`:

```
# Abort the program, displaying an error
die() {
   echo " ERROR: $*" >&2
   exit 1
}

# arg1: full path of configuration file to read
read_conf() {
   if [ -f "$1" ] ; then
      source "$1" || die "Could not source $1"
      echo " Processed Config: $1"
   fi
   return 0
}
```

These functions are very simple but perform useful functions and help clean up the main code. The `die` function displays a properly formatted error message and exits the script (with an unsuccessful return code). The `read_conf` function checks for a configuration file, sources the file (if found), and aborts if it finds the file but cannot read it.

6.3.3.2.1 The do_template Function

The following `do_template` function also needs to be placed in the function file that is sourced by the `configure_system` script:

```
# arg1: full path of local source file
# arg2: full path of local destination file
do_template() {
   local src="$1"
   local dest="$2"
   [ -z "$src" -o -z "$dest" ] &&
      die "do_template called with empty parameter(s) ($*)"
```

```
   # Process the template
   sed 's/\\/\\\\\\\\/g; s/"/\\\\"/g;' "$src" | while read line; do
      eval echo "\"$line\"";
   done > "$dest" || die "Could not process template $src"
   echo " Processed Template: $(basename "$src")"
}
```

One part of this function definitely requires an explanation. You saw that I had to use the `sed` command to escape backslashes and double quotes. Using this is necessary if I want to preserve these characters throughout the `echo` command, followed by the `eval` command. Additionally, since the `sed` command also requires backslashes to be escaped, the actual `sed` substitution pattern contains eight backslashes. This is the first part of the `sed` expression (`s/\\/\\\\\\\\/g`). Here is what happens at the various stages:

Original `\`
After `sed` `\\\\`
After `echo` `\\`
After `eval` `\`

The translation of the double quotes should also seem very similar. This is the second half of the two-part `sed` command (`s/"/\\\\"/g`). The journey of the double quote character through all of these steps is as follows:

Original `"`
After `sed` `\\"`
After `echo` `\"`
After `eval` `"`

6.3.3.2.2 The push_file Function

The `push_file` function is provided in this section. It also needs to be placed in the function file that is sourced by the `configure_system` script that I will provide you with later in this chapter.

The `push_file` function is the most complex of the helper functions. It has to parse information out of the source file, remove that metadata, transfer the file, and possibly set ownership and permissions.

As mentioned earlier in this chapter, any file that is being pushed to the remote system may have any number of the following lines containing metadata to be used by the script:

```
__:LOCATION=/etc/resolv.conf
__:PERMS=0755
__:USERGROUP=root.root
```

In fact, the script is written so that any line starting with __: will be evaluated (to set these variables) and removed from the pushed file. If, for some reason, one of your files is supposed to have a line that begins with that sequence, you will need to modify the script to use a different sequence than the one used in the example.

NOTE The line numbers in this code and other code sections in this chapter are for display purposes only and are not part of the actual program code.

```
1  # arg1: full path of local source file
2  # arg2: hostname of remote system
3  # arg3: [optional] destination file on remote system
4  push_file() {
5     local src="$1"
6     local host="$2"
7     local dest="$3"
8     [ -z "$src" -o -z "$host" ] &&
9        die "push_file called with missing parameter(s) ($*)"
10
11    # Read configuration items from file
12    eval `sed -n 's/^__://p' "$src"`
13
14    # Override location if necessary
15    [ -n "$dest" ] && LOCATION="$dest"
16
17    # Push the file now
18    grep -v '^__:' "$src" | ssh -T "root@$host" "cat > $LOCATION" ||
19       die "Could not place remote file $LOCATION"
20
21    # Set the permissions, if necessary
22    [ -n "$PERMS" ] && ssh "root@$host" "chmod $PERMS $LOCATION" ||
23       die "Could not set permissions on remote file $LOCATION"
24
25    # Set the ownership, if necessary
26    [ -n "$USERGROUP" ] && ssh "root@$host" \
27       "chown $USERGROUP $LOCATION" ||
28       die "Could not set permissions on remote file $LOCATION"
29
```

```
30     [ -z "$dest" ] && {
31        # Only produce output if third arg was not given
32        # (third arg is given when pushing scripts to execute)
33        echo "   Pushed File: $LOCATION"
34     }
35  }
```

Line 12: The `sed` command is used only to print lines in the file that begin with the special `__:` sequence. In addition, that sequence is stripped from the beginning of the lines, leaving a series of commands suitable for being evaluated by the shell. This output is then processed by the `eval` so that the variables are set. So, for example, the line `__:PERMS=0755` is converted by `sed` to `PERMS=0755`, which is evaluated by the `eval` command, which causes the `PERMS` environment variable to be set with the value `0755`.

Line 15: If a third parameter was provided to the function, the `dest` variable will not be empty, and the `-n` test will succeed. The `LOCATION` variable will be set to that value whether it was specified in the file or not.

Line 18: The inverse (`-v`) `grep` command is used to remove the metadata lines from the file. The output stream is sent through SSH to the remote system. Although `scp` could have been used, it would have required another temp file. The `ssh` command works fine for transferring files (as long as you use the `-T` switch to disable tty allocation). On the remote side, the `cat` command takes its `stdin` and outputs it to `stdout`, which is redirected to the destination file.

6.3.3.3 The configure_system Script

This shell script configures a single remote host. It uses the host directory specified (such as `/usr/local/etc/hosts`) to determine the system's associated groups. It uses all of the files and directories in the `/usr/local/etc/sysconfig` directory as described in section 6.3.1. It also requires all of the functions shown in the previous section to be in the specified file (`/usr/local/sbin/configure.functions`).

```
1  #!/bin/bash
2
3  # The base directory
4  host_dir='/usr/local/etc/hosts'
5  base_dir='/usr/local/etc/sysconfig'
6  source '/usr/local/sbin/configure.functions'
7
```

```
8  # Specify any necessary options for ssh here
9  alias ssh="/usr/bin/ssh"
10
11 [ $# -lt 1 ] && die "Usage: $0 host"
12 host="$1"
13 echo "Configuring host $host:"
14
15 # We need a safe local and remote temp file
16 localtemp=`mktemp /tmp/sysconfig-XXXXXX` ||
17    die "Could not create local temp file"
18 remotetemp=`ssh $host 'mktemp /tmp/sysconfig-XXXXXX'` ||
19    die "Could not create remote temp file"
20
21 # First, retrieve group list for the host
22 cd "$host_dir" || die "Host directory $host_dir not found"
23 groups="$(grep -l "^$host$" *)"
24
25 # Read settings for this host
26 read_conf "$base_dir/conf/all"
27 for group in $groups ; do
28    read_conf "$base_dir/conf/groups/$group"
29 done
30 read_conf "$base_dir/conf/systems/$host"
31
32 # Now, push out any necessary files
33 for dir in all $groups ; do
34    for file in "$base_dir/files/$dir/"* ; do
35       if [ -f "$file" ] ; then
36          push_file "$file" "$host"
37       fi
38    done
39 done
40
41 # Next, push out any necessary templates
42 for dir in all $groups ; do
43    for file in "$base_dir/templates/$dir/"* ; do
44       if [ -f "$file" ] ; then
45          do_template "$file" "$localtemp"
46          push_file "$localtemp" "$host"
47       fi
48    done
49 done
50
```

```
51 # Finally, execute any necessary scripts
52 for dir in all $groups ; do
53    for file in "$base_dir/scripts/$dir/"* ; do
54       if [ -f "$file" ] ; then
55          push_file "$file" "$host" "$remotetemp"
56          ssh "root@$host" "chmod u+x $remotetemp ; $remotetemp" ||
57             die "Execution of script $(basename "$file") failed"
58          echo "   Executed Script: $(basename "$file")"
59       fi
60    done
61 done
62
63 # Clean up
64 rm -f "$localtemp"
65 ssh "root@$host" "rm -f $remotetemp"
66
67 exit 0
```

Line 16: The `mktemp` command is used to create both a local and remote temporary file, using `/tmp/sysconfig-XXXXXX` as its pattern (the `X` characters are replaced by random characters). The `mktemp` command returns the name of the actual file created on `stdout`. It would even be better to put these temporary files in a directory that normal users cannot access.

Line 23: The `-l` switch for the `grep` command causes a list of filenames with matching lines to be printed. This list is the list of groups associated with the specified host.

Line 26: Each necessary configuration file is read, in order, by the `read_conf` command. If a setting is defined in more than one file, the last value read will be used.

Line 33: Each file in the `all/` directory, as well as each file in any appropriate group directories, is pushed out to the remote system using the `push_file` function.

Line 42: This loop is just like the files loop, but for templates. Each template is first processed using the `do_template` function, and then it is pushed to the remote system using the `push_file` function.

Line 52: This loop is the same as the files and templates loop. Each local file is a script or other executable program. The file is pushed out as the remote temporary file, made executable, and then executed.

6.3.4 Exploring Example Configuration File Creation

The actual configuration script might be a relatively minor part of your custom configuration system. Depending on how many configuration files you need to push to your various systems, you can end up spending most of your time creating the actual configuration files and templates. Unfortunately, most UNIX variants have their own method for configuring much of the system. Applications, of course, also have widely varied configuration methods, file formats, and so on, so I will not be able to provide a set of templates that will work with your specific set of systems. I will, however, provide a couple simple examples that should help you start creating your own template system.

6.3.4.1 The /etc/resolv.conf File

For our first example, we will consider the `/etc/resolv.conf` file, which is found on just about every UNIX system. A typical system will have the following `/etc/resolv.conf` file:

```
nameserver 192.168.2.2
nameserver 192.168.2.3
search mydomain.com
```

You would place the following lines in the `conf/all` file:

```
SEARCH_DOMAIN="mydomain.com"
NAMESERVER1="192.168.2.2"
NAMESERVER2="192.168.2.3"
```

Let's say that all desktop machines should also have `dev.mydomain.com` in their search domains. You would put this line in `conf/groups/desktop`:

```
SEARCH_DOMAIN="${SEARCH_DOMAIN} dev.mydomain.com"
```

Finally, you would create the following template in `templates/all/resolv.conf`:

```
__:LOCATION=/etc/resolv.conf
__:PERMS=0755
__:USERGROUP=root.root
nameserver ${NAMESERVER1}
nameserver ${NAMESERVER2}
search ${SEARCH_DOMAIN}
```

6.3.4.2 The /etc/mail/access File

Let's say that you run sendmail on several hosts in your network. Each of those hosts is in the `sendmail` group. Let's say all of your IP addresses are in the `192.168.0.0/255.255.0.0` subnet (which contains addresses from `192.168.0.0` through `192.168.255.255`). You want to place `/etc/mail/access` on each of these hosts to allow mail to be received from any other host in your network. Since there is no variable information in this particular example, you use a static file. The `files/sendmail/access` file should be as follows:

```
__:LOCATION=/etc/mail/access
__:PERMS=0755
__:USERGROUP=root.root
localhost.localdomain                    RELAY
localhost                                RELAY
127.0.0.1                                RELAY
192.168                                  RELAY
```

In this case, however, the binary `/etc/mail/access.db` file needs to be rebuilt after the `access` is updated. So, you need a script to be run on each remote system. This script should be placed in `scripts/sendmail/update_db.sh`:

```
#!/bin/bash

cd /etc/mail
make >/dev/null 2>&1
exit 0
```

This script will be run on each host in the `sendmail` group after the `/etc/mail/access` file is placed on the system. The script shown always exits with a positive return value. You may want to remove the `exit` command so that the script exits with the return value of the `make` command. This way, if the `make` fails, the configuration process terminates, and you can determine what went wrong. Of course, this might not be desirable if you run the configuration script automatically.

6.4 Configuring Systems with GNU cfengine

GNU's cfengine satisfies all of the requirements mentioned in this chapter and many more. In addition to system configuration, it can also perform a range of monitoring and maintenance tasks that I will discuss in future chapters.

Although it can be a bit overwhelming at first, installing and configuring cfengine is going to be easier than making your own system in many cases. The

larger your network of systems, and the more diverse they are, the greater the advantages of using cfengine. Since many environments tend to grow bigger and faster than you expect, cfengine may be worth the effort in most cases.

That being said, there are still some situations in which you might be better off using a custom approach. A stand-alone network appliance is a good example. Although cfengine could indeed do the job, it might be overkill when you have only one machine. You might only need to do a few things that could be done with custom scripts rather quickly. The important thing is to evaluate your available options and choose the best one for your particular situation.

The single biggest drawback (as well as one of the best features) of cfengine is its complexity. It would probably take an average system administrator a week to get cfengine operating fully on only a few systems. Of course, once you have accomplished that feat, expanding its operations to hundreds of systems is relatively painless.

The most current version of cfengine at the time of this writing was 2.0.4 and it is the version used within this chapter. The 2.*X* series has quite a few changes from the 1.*X* series, so you will see differences between the examples in this book and older versions of cfengine. If you are just getting started, I would recommend that you use a newer version of cfengine.

6.4.1 cfengine Overview

In this section, I provide a brief overview of cfengine, its files, its methods, and its most important concepts.

6.4.1.1 cfengine Concepts

The GNU cfengine program was designed to save time and reduce headaches in the long run. It may take some time to set up and configure, but you will be happier when everything has been said and done. At least at first, performing new tasks with cfengine will take longer than performing the same task manually. But, when you upgrade the operating system and loose some change, you will be glad you used cfengine because it will simply perform the change again. Or, when you realize a few other systems need the same change, you can use cfengine to make this happen in seconds (by adding the new systems to the appropriate class).

If you made the change manually, on the other hand, it might take some time before you even notice that the change needs to be made again. Once you notice, you'll have to make the change manually all over again—that is, of course, if you remember how you did it the last time. If ten new systems need a specific change, you might spend an hour changing each system yourself, whereas cfengine could have just done it for you.

One major design principle for cfengine was that it should operate using one central set of configuration files. This enables the same set of configuration files to be executed on every host on the network in their original form. Each host also transfers the configuration files from the server (if possible) before each run. As long as you make all the changes in that central set of configuration files, then everything else will happen automatically. You will not have any more manual changes to systems that you may forget about. You will no longer have to use special scripts for special systems and/or scripts that have so many conditionals (based on hostname, operating system, etc.) that they have become unreadable and difficult to maintain. Perhaps most importantly, this central set of configuration files documents every change you have done to every system. If you put a few comments in the files along with the commands, you will not only document what you have done, but also document *why* you did it.

For some tasks, cfengine abstracts the actual desired action from the technical specifics of the underlying operating system. For other tasks (namely editing files), cfengine provides a high-level language that allows you to exactly specify the actual modifications.

For other tasks, cfengine does not provide any native support. It does, however, allow you to execute external scripts based on a system's class membership. When possible, you should use the internal commands provided by cfengine. If you do not, you can use custom shell and Perl scripts, but you should still get cfengine to execute them on your behalf.

Once you decide to use cfengine, it is in your best interest to use it for as many tasks as possible. This usually means that you will need to change your habits because it will always be tempting to just "fix it real quick" instead of going through the proper cfengine process. However, once you switch fully to cfengine, you will enjoy many benefits:

- A standardized, centralized configuration that all hosts on your network can use.
- Systems that can be classified using a variety of methods and classes that can specify certain behavior.
- Changes that when made to systems are recorded and performed again, if necessary.
- Systems that may have intermittent uptime or network connectivity but will eventually make any necessary changes.

6.4.1.2 Push vs. Pull

Yet another advantage to using cfengine is that it pulls from a server instead of pushing from the master system. This doesn't make a big difference when you have a local set of servers that are reliable and usually up and running; if, however, your systems are spread out over an unreliable network and/or may not always be running, the pull method is much more reliable.

For example, if some systems can boot to either Linux or Windows, they can pull from the server whenever they are running in Linux. If you used a push technique instead, the system might get neglected if it was running Windows at the time the push was attempted. Here is another example: you might have UNIX running on one or more laptops that are not always connected to the network. A system like this might never be updated using the push method, because it would have to be connected to the network at the exact time a push occurs. With the pull method, the laptop would automatically pull the configuration changes the next time it could contact the configuration server.

There is, of course, no reason why the custom method just discussed couldn't be adapted to use the pull method. If you need to use the pull method, but you don't want to use cfengine for some reason, try using the custom scripts earlier in this chapter—they would be a good place to start. Either way, you will be able to make changes on a server and have each client pull those changes from the server at their next opportunity.

Although cfengine typically pulls from a server and executes every hour, it also supports the ability to force updates to all or any subset of the systems on demand. Obviously, you will find this very useful when you are performing mission-critical bug fixes (e.g., something else you did messed up a system or two and you need to fix them very quickly).

You can also run cfengine directly on each system by logging in and manually running the `cfagent` command. I think cfengine follows a good theory in system administration automation: the more ways you can initiate changes to a system, the better—as long as they are done in the same way. In other words, cfengine provides several methods for updating each system, but all of them use the same configuration files and operate exactly the same way (once initiated).

6.4.1.3 The Components of cfengine

The cfengine suite consists of several compiled programs. I will not list all of these programs here, only the ones that we will discuss in this book. These programs are typically installed in a location that is common to all systems (such as an NFS-mounted `/usr/local`). Alternatively, they can be directly installed on each system by a package management system. In either case, the important binaries (`cfagent`,

cfexecd, and cfservd, described momentarily) are copied to a local directory such as /var/cfengine/bin/ to make sure that they are always available, even when there may be problems with the network.

cfagent: The autonomous configuration agent (the heart of the program). This command can be run manually (on demand), by cfexecd on a regular basis, and/or by cfservd by remote command.

cfservd: The file transfer and remote activation daemon. This must be run on any cfengine file servers and on any system that you would like to be able to execute cfagent remotely.

cfexecd: The execution and reporting daemon. This is either run as a daemon or run as a regular cron job. In either case, it handles running cfagent and reporting its output.

cfkey: Generates public/private key pairs and needs to be run only once on every host.

cfrun: This command needs to be run from a server that will contact the clients (through cfservd) and tell them to execute cfagent.

For any given command, you can see a summary of its options by using the-h command-line option. When running a command, you can always specify the -v switch to see more (or in many cases, any) detail. When debugging a program, you should use the -d2 switch to view debugging information (and, for daemons, the -d2 switch prevents it from detaching from the terminal).

6.4.1.4 cfengine Directory Structure

The binaries need to be installed in a directory mounted on every host or they need to be installed independently on each host. Everything cfengine uses during its normal operation is located under the /var/cfengine/ directory. Its contents are as follows:

bin/: Important binaries (cfagent, cfexecd, and cfservd) are copied here to ensure that they are available when needed.

inputs/: This is the standard location for all of the configuration files cfengine needs. We will be using three files from this directory: cfagent.conf, cfservd.conf, and update.conf.

outputs/: This is where the output files from each run cfexecd performs are placed.

ppkeys/: This is where this system's public and private keys, as well as other systems' public keys, are located.

6.4.1.5 cfengine Configuration Files

Each system must have a minimal number of configuration files. These are found in the `/var/cfengine/inputs/` directory on each system, but they should be maintained in a central location.

`update.conf`: This file must be kept simple. It is always parsed and executed by `cfagent` first. Its main job is to copy the set of configuration files from the server. If any of the other configuration files contain an error, this file should still be able to update files so that the next run will be successful. If there is an error in this file, you will have to manually fix any affected systems.

`cfagent.conf`: This file contains the guts of your automation system. It contains all of the actions, group declarations, and so on. Although our examples in this book are simple enough to neatly fit into one file, this file could include any number of separate files to make things simpler (using the `import` section).

`cfservd.conf`: This, as may be obvious, is the configuration file for the `cfservd` daemon. It defines which hosts can remotely execute `cfagent` and which remote hosts can transfer which files.

The master copy of the configuration files should be managed with a source control system like CVS. This way you have a record of any changes made and you have the ability to revert to an older configuration file if you introduce a problem into the system.

6.4.1.6 Identifying Systems with Classes

The concept of classes is at the heart of cfengine. Each system belongs to one or more classes. Or, if you think of it another way, many classes are defined each time `cfagent` runs based on a variety of different kinds of information. Each action in the configuration file can be limited to only certain classes. So, any given action could be performed only on one host, or only on hosts that are running a specific operating system, or on every host. cfengine uses both built-in and custom classes, both of which will be discussed within this section.

6.4.1.6.1 Predefined Classes

The host itself determines many of the classes that are defined—its architecture, hostname, IP address(es), and operating system. Several classes are also defined based on the current date and time.

To determine which standard classes are defined on any give system, run this command:

```
# cfagent -p -v | grep Defined
Defined Classes = ( any redhat redhat_7 redhat_7_2 Sunday Hr16 Min58
Min55_00 Q4 Hr16_Q4 Day11 August Yr2002 linux kaybee_org org kaybee
32_bit linux_2_4_9_31smp i686 linux_i686 linux_i686_2_4_9_31smp
compiled_on_linux_gnu ipv4_10 10_1_1_1 10_1_1 ipv4_10_1_1_1
ipv4_10_1_1 ipv4_10_1 )
```

As you can see, there are quite a number of predefined classes for my system. They can be divided into a few categories:

Operating System: `redhat redhat_7 redhat_7_2`

Kernel: `linux linux_2_4_9_31smp`

Architecture: `i686 linux_i686 linux_i686_2_4_9_31smp`

Hostname: `kaybee_org kaybee org`

IP Address: `10_1_1 10_1_1_1 ipv4_10 ipv4_10_1 ipv4_10_1_1 ipv4_10_1_1_1`

Date/Time: `Sunday Hr16 Min58 Min55_00 Q4 Hr16_Q4 Day11 August Yr2002`

Every system is a member of the `any` class, for obvious reasons. As you can see, cfengine provides quite good granularity with its default class assignments. I cannot possibly list all the classes that could be assigned to your systems, so you will have to check the list on each of your systems (or, at least, each type of system on your network).

The time-related classes probably require some additional explanation. The `Min55_00` class specifies the current five-minute range. The `Q4` class is always set in the last quarter of the hour. The `Hr16_Q4` class says we are currently in the last quarter of the 16th hour.

6.4.1.6.2 Custom Classes

Custom classes are defined in the `classes` section of the `cfagent.conf` configuration file. Here is an example:

```
classes:
   # Check to see if X11R6 is installed
   X11R6 = ( '/usr/bin/test -d /usr/X11R6' )

   # Mail servers must be explicitly defined
   mail = ( mail1 mail2 )
```

```
# DNS and web servers are obvious by their configuration files
dns = ( '/usr/bin/test -f /etc/named.conf' )
web = ( '/usr/bin/test -f /etc/httpd/conf/httpd.conf' )

# Any critical servers are a member of this class
critical = ( mail dns web )
```

When a class definition contains a quoted string, that is a command to be executed. If it returns an exit code of `0`, then this system will be part of that class.

Class definitions can (and usually do) list other classes. If a system is a member of any of the listed classes, then it is also a member of the new class.

Some cfengine commands can define new classes in certain situations. If, for example, a particular drive is too full, a class can be defined (and a warning can also be printed). Other parts of the configuration file may take further actions if that class is defined. I explore this further later in this chapter.

6.4.1.7 More Information About GNU cfengine

GNU cfengine can be downloaded from its web site `http://www.cfengine.org/`. The web site includes a tutorial, a comprehensive reference manual, mailing lists, and archives.

You should also examine the large number of sample configuration files that are included with the cfengine distribution. These sample files, when combined with the web-based reference manual, should get you going on even the most advanced uses of cfengine.

6.4.2 Basic Setup

Within this section, I illustrate and discuss a simple cfengine setup that will provide a good framework for customization and expansion. These simple configuration files will not make many changes to your systems, but they will still show some of the power of cfengine.

This simple setup includes one central server and one other host. With cfengine, all hosts are set up identically (even with only slight differences on the server), so this example could be extended to any number of systems. I would recommend, though, that you start out with just two systems. Once you get cfengine up and running on those systems, it is easy enough to expand the system to other hosts.

6.4.2.1 Setting Up the Network

Before starting with cfengine, you should make sure that your network is properly prepared. It is difficult, if not impossible, to use cfengine with dynamic IP addresses. Even if you use the Dynamic Host Configuration Protocol (DHCP) to assign addresses to some or all of your systems, it should always assign the same IP address to systems that will be controlled with cfengine. In other words, it doesn't matter which method you use to assign the IP addresses, as long as each system that is going to be managed has a consistent IP.

The next task is to make sure your Domain Name System (DNS) is properly configured for your hosts. Although you can always work around DNS problems, it is best to have properly configured DNS for your testing hosts. Each host should have a hostname, and a DNS lookup of that hostname should return that host's IP address. In addition, if that IP address is looked up in DNS, the same hostname should be returned.

If this setup is not possible, I recommend that you add every host to the `/etc/hosts` file on every system. If you are using a multihomed host, you have to pay attention to which IP address will be used when your host is communicating with other cfengine hosts.

6.4.2.2 Running Necessary Processes

In the most simplistic setup, you can use cfengine by running `cfagent` on each system manually. You will, however, benefit more from running one or two daemons on each system.

6.4.2.2.1 The cfexecd Daemon

Although you could, theoretically, only run `cfagent` on demand, it is better to run it automatically on a regular basis. This is when `cfexecd` comes in handy; it runs as a daemon and executes `cfagent` on a regular, predefined, schedule. This schedule can be modified by adding time classes to the `schedule` setting in the `control` block in `cfagent.conf`. The default setting is `Min00_05`, which means `cfagent` will run in the first five minutes of every hour. To run twice per hour, for example, you could place the following line in the `control` section of `cfagent.conf`:

```
schedule = ( Min00_05 Min30_35 )
```

The `cfexecd` daemon does not have its own configuration file, but it does use this setting out of `cfagent.conf`.

You can also run `cfexecd` on a regular basis using the system's cron daemon. The following entry could be added to the system crontab (usually `/etc/crontab`) to execute (and report) `cfagent` every hour:

```
0 * * * * root /usr/local/sbin/cfexecd -F
```

The `-F` switch tells the `cfexecd` not to go into daemon mode because it is being run by cron.

For the ultimate in reliability, run `cfexecd` as a daemon and also run it from cron (maybe once per day). You can then, in `cfagent.conf`, check for the crontab entry and check to see if the `cfexecd` daemon is running. The following lines, if placed in `cfagent.conf`, perform these checks and correct any problems:

```
editfiles:
   { /etc/crontab
      AppendIfNoSuchLine "0 * * * * root /var/cfengine/bin/cfexecd -F"
   }

processes:
   "cfexecd" restart "/var/cfengine/bin/cfexecd"
```

With this technique, if either of the methods is not working properly, the other method ultimately repairs the problem.

6.4.2.2.2 The cfservd Daemon

The `cfservd` daemon is not required on all systems. It needs to run on any cfengine file servers, which, in our case, is the central configuration server only. It also allows you to remotely execute `cfagent` from other systems. If you want this functionality, then `cfservd` needs to be running on every system. In either case, you should always check to make sure it is running with the following command in `cfagent.conf`:

```
processes:
   "cfservd" restart "/var/cfengine/bin/cfservd"
```

6.4.2.3 Creating Basic Configuration Files

These configuration files need to be placed in the master configuration directory on the configuration server (as explained in the next section). They are common files and will be used in their exact original form on every server in your network.

6.4.2.3.1 Example cfservd.conf

This is the configuration file for the `cfservd` daemon. It allows clients to transfer the master set of configuration files and also allows `cfagent` to be remotely executed using `cfrun`. Obviously only one system needs to allow access to the central configuration files (the server), but having `cfservd` allow access to those files does not hurt anything on other systems (because they don't have the files there to copy). All systems, however, can benefit from allowing remote `cfagent` execution, since it allows you to execute `cfagent` on demand from remote systems.

So, the following `cfservd.conf` can be used on all systems, which is what we want:

```
control:
   domain = ( mydomain.com )
   AllowUsers = ( root )
   cfrunCommand = ( "/var/cfagent/bin/cfagent" )

admit:
   /usr/local/var/cfengine/inputs *.mydomain.com
   /var/cfagent/bin/cfagent       *.mydomain.com
```

The `cfrunCommand` setting specifies the location of the `cfagent` binary to be run when a connection from `cfrun` is received. The `admit` section is very important because it specifies which hosts have access to which files. You must grant access to the central configuration directory and the `cfagent` binary. You also need to grant access to any other files that clients need to transfer from this server.

6.4.2.3.2 Basic update.conf

The `update.conf` file must be kept as simple as possible and should rarely, if ever, be changed. This file is parsed and executed by `cfagent` before `cfagent.conf`. If you put out a bad `cfagent.conf`, the next time the clients execute `cfagent` they get the new version because their `update.conf` file is still valid. Distributing a bad `update.conf` would not be a very good idea, so I recommend thoroughly testing any changes before you place the file in the central configuration directory.

Again, this file is run on every host, including the server. The `cfagent` command is smart enough to copy the files locally (instead of over the network) when running on the configuration server. Several variables are defined in the `control` section and then used in the `copy` section. Variable substitution can be accomplished with either the `$(var)` or `${var}` sequences.

```
1  control:
2   actionsequence  = ( copy tidy )
3   domain          = ( mydomain.com )
4   workdir         = ( /var/cfengine )
5   policyhost      = ( server.mydomain.com )
6   master_cfinput  = ( /usr/local/var/cfengine/inputs )
7   cf_install_dir  = ( /usr/local/sbin )
8
9  copy:
10       $(cf_install_dir)/cfagent    dest=$(workdir)/bin/cfagent
11                                              mode=755
12                                              type=checksum
13
14       $(cf_install_dir)/cfservd    dest=$(workdir)/bin/cfservd
15                                    mode=755
16                                    type=checksum
17
18       $(cf_install_dir)/cfexecd    dest=$(workdir)/bin/cfexecd
19                                    mode=755
20                                    type=checksum
21
22       $(master_cfinput)            dest=$(workdir)/inputs
23                                    r=inf
24                                    mode=644
25                                    type=binary
26                                    exclude=*.lst
27                                    exclude=*~
28                                    exclude=#*
29                                    server=$(policyhost)
30 tidy:
31       $(workdir)/outputs pattern=* age=7
```

Line 5: The string `server.mydomain.com` should be replaced with the hostname of your configuration server.

Line 6: This is the directory on the master configuration server that contains the master configuration files.

Line 7: This is the location, on every server, of the cfengine binaries.

Line 11: This specifies that the source directory should be copied recursively to the destination directory with no limit on the recursion depth.

Line 13: This option is misleading at first. It specifies that any local file should be compared byte-by-byte with the master copy to determine if an update is required.

Line 17: This option causes the files to be retrieved from the specified server.

Line 31: This command in the `tidy` section removes any files in the `outputs/` directory that have not been accessed in the last seven days.

The permissions (modes) on each file are checked on each run even if the file already exists.

6.4.2.3.3 Framework for cfagent.conf

This is the meat of the cfengine configuration. Hopefully, any change you need to make on any system will be done using this file. The sample `cfagent.conf` given here is very simple for testing purposes, and more advanced `cfagent.conf` uses will be discussed further in section 6.4.4.

If you call any scripts from cfengine and those scripts produce any output, that output will be displayed (when executed interactively) or logged and emailed (when executed from `cfexecd`). Since it is typical to execute `cfagent` every hour, any scripts should only produce output if there is a problem or if something changed and the administrator needs to be notified.

```
1  control:
2     actionsequence  = ( files directories tidy disable processes )
3     domain          = ( mydomain.com )
4     timezone        = ( EDT EST )
5     access          = ( root )
6     # Where cfexecd sends reports
7     smtpserver      = ( mail.mydomain.com )
8     sysadm          = ( root@mydomain.com )
9
10 files:
11    /etc/passwd mode=644 owner=root action=fixall
12    /etc/shadow mode=600 owner=root action=fixall
13    /etc/group  mode=644 owner=root action=fixall
14
15 directories:
16    /tmp mode=1777 owner=root group=root
17
18 tidy:
19    /tmp recurse=inf age=7 rmdirs=sub
20
```

```
21  disable:
22     /root/.rhosts
23     /etc/hosts.equiv
24
25  processes:
26     "cfservd" restart "/var/cfengine/bin/cfservd"
27     "cfexecd" restart "/var/cfengine/bin/cfexecd"
```

Line 2: The `actionsequence` command is very important and easy to overlook. You must list each section that you wish to process in this variable. If you add a new section but forget to add it to this list, it will not be executed.

Line 4: cfengine will make sure the system is configured with one of the time zones in this list.

Line 10: This section checks the ownership and permissions of a few important system files and fixes any problems it finds.

Line 15: This section checks the permissions on the `/tmp/` directory and fixes them, if necessary. It also creates the directory, if necessary.

Line 18: This section removes everything that has not been accessed in the past seven days from the `/tmp/` directory. It only removes subdirectories of `/tmp/` and not the directory itself.

Line 21: These files are disabled for security reasons. They are renamed if they are found. If they are executable, the executable bit is unset.

Line 25: This section verifies that the `cfservd` and `cfexecd` daemons are running and starts them if they are not.

6.4.2.4 Creating the Configuration Server

The configuration server contains the master copy of the cfengine configuration files. It also processes that configuration file on a regular basis just like all of the client systems. The server must run a properly configured `cfservd` so that the client systems can retrieve the master configuration from the system.

The configuration server needs to have a special place to keep the master cfengine configuration files. In this example, that directory is `/usr/local/var/cfengine/inputs/`. It could be any directory, but not `/var/cfengine/inputs/` because the master host copies the files to that directory when executing, just like every other host.

Like all systems, the server should also run `cfexecd` either as a daemon or from cron (or, even better, both).

6.4.2.4.1 Generating Server Keys

You need to run `cfkey` on the server system to create its public and private key files. These files will be in the `/var/cfengine/ppkeys/` directory and will be named `localhost.priv` and `localhost.pub`. You then need to copy the `localhost.pub` to a new file in the same directory. This file should be called `root-10.1.1.1.pub`, assuming your IP address is `10.1.1.1`. This is only necessary to allow the `cfrun` command to connect to itself, but it is good to do anyway.

The server also needs each client's public key in the appropriate file, based on the client's IP address as described in the next section.

6.4.2.5 Preparing the Client Systems

Each client system is relatively simple to configure. Once you install the actual cfengine binaries, you need to generate and copy the appropriate public keys (as discussed in this section). You also need to manually copy the `update.conf` file from the master server and place it in `/var/cfengine/inputs/`. Once this file is in place, you should manually run `cfagent` to download the remaining configuration files and complete system configuration.

Each client should run `cfexecd` either as a daemon or from cron. You probably want to run `cfservd` on each client as well to allow remote execution of `cfagent` using `cfrun`. Assuming this is already configured in the `cfagent.conf` file on the server, these daemons will be started after the first manual execution of `cfagent`.

6.4.2.5.1 Generating Client Keys

You need to run `cfkey` on each client system. This creates `localhost.priv` and `localhost.pub` in `/var/cfengine/ppkeys/`. You then need to copy the central server's public key to the client. If the server's IP address is `10.1.1.1`, then you should copy its public key to `root-10.1.1.1.pub` in the `/var/cfengine/ppkeys/` directory on the client.

Finally, you need to copy the client's public key (`localhost.pub`) to the server's `/var/cfengine/ppkeys/` directory. Again, this file should be named according to the client's IP address (if its IP address is `10.2.2.2`, then the file should be named `root-10.2.2.2.pub` on the central server. This can be done manually, or it can be done automatically from your initial system configuration script (as discussed in section 5.3).

6.4.3 Debugging cfengine

When you are trying to get cfengine up and running, you will probably run into a few problems. Network problems are common. This includes the processes of transferring configuration files from the master server and initiating `cfagent` execution on remote systems with `cfrun`.

For any network problems, you should run both the server (`cfservd`) and the client (`cfrun` or `cfagent`) in debugging mode. You can accomplish this by using the `-d2` command-line argument. For `cfservd`, this switch not only provides debugging output, but it also prevents the daemon from detaching from the terminal.

When you are trying new things in your `cfagent.conf`, you should always try it with the `--dry-run` switch to see what it would do without making any actual changes. The `-v` switch is also very useful if you want to see what steps are being taken by `cfagent`. If it is not doing something you think it should be doing, the verbose output will probably tell you why.

If you are making frequent changes or trying to get a new function to work properly, you probably want to be able to run `cfagent` repeatedly on demand. By default, `cfagent` will not do anything more frequently than once per minute. This helps prevent both intentional and accidental denial-of-service attacks on your systems.

You can eliminate this feature for testing purposes by placing this line in the `control` section of `cfagent.conf`:

```
IfElapsed = ( 0 )
```

It is also helpful to only run a certain set of actions by using the `--just` command-line option. For example, to check only on running processes, you can run the command **`cfagent --just processes`.**

6.4.4 Creating Sections in cfagent.conf

There are, as of cfengine version 2.0.4, 24 possible sections in `cfagent.conf`. I only cover some of these sections in this chapter. Other sections will be covered later in this book, while some will not be covered at all. In addition, some of the sections can be quite powerful and will not be fully explored in this book. For additional information, refer to the comprehensive reference manual that can be read on the cfengine web site at `http://www.cfengine.org/`.

Every section can contain one or more class specifications. For example, the `files` section could be:

```
files:
   /etc/passwd

   any::
      /etc/group
   redhat::
      /etc/redhat-release
   solaris::
      /etc/vfstab
```

Both `/etc/passwd` and `/etc/group` will be processed on all hosts (because the default group is `any` when none is specified). In addition, the `/etc/redhat-release` file will be checked only on systems running Red Hat Linux, and the `/etc/vfstab` will be checked only if the operating system is Sun's Solaris.

The period (`.`) can be used to "and" groups together, whereas the pipe character (`|`) can be used to "or" groups together. The exclamation character (`!`) can invert a class and parentheses (`(/)`) can be used to group classes. Here is a complex example:

```
files:
   (redhat|solaris).!Mon::
      /etc/passwd
```

In this case, the `/etc/passwd` file will only be checked if the operating system is Red Hat Linux or Sun's Solaris and today is not a Monday.

6.4.4.1 Using Classes in cfagent.conf

The `classes` section can be used to create user-defined classes as described in section 6.4.1.6.2. A system's class membership can be determined by a shell command as follows:

```
classes:
   X11R6 = ( '/usr/bin/test -d /usr/X11R6' )
```

If the command returns `true` (exit code 0), this machine will be a member of that class (for the current run). Classes can also be defined to contain specific hosts or any hosts that are a member of another existing class, as follows:

```
classes:
   critical = ( host1 host2 web_servers )
```

Here are a few more possibilities that could be placed in the `classes` section:

```
classes:
   notthis = ( !this )
   ip_in_range = ( IPRange(129.0.0.1-15) )
   ip_in_range = ( IPRange(129.0.0.1/24) )
```

6.4.4.2 The directories Section

The `directories` section checks for the presence of one or more directories. Here is an example:

```
directories:
   /etc mode=0755 owner=root group=root syslog=true
      inform=true
   /tmp mode=1777 owner=root group=root define=tmp_created
```

If either directory does not exist, it will be created. The permissions and ownership will also be checked and corrected, if necessary. In this example, the administrator will be informed (through mail or the terminal) and a syslog entry will be created if the `/etc/` directory does not exist, or if it has incorrect permissions and/or ownership.

For the `/tmp/` directory, the class `tmp_created` is defined if the directory was created. You could then use this class in another section of the configuration file to perform certain additional tasks if the directory was created.

6.4.4.3 The disable Section

The `disable` section causes cfengine to disable any number of files. This simple section disables two files that you probably would never want around because they are used to allow access to the `root` account via Remote Shell (RSH) using only the source IP address as authentication:

```
disable:
   /root/.rhosts
   /etc/hosts.equiv
```

If either of these files exist, they will be disabled by being renamed with a `.cfdisabled` extension. The permissions will also be changed to `0600`, so this command is also suitable for disabling executables. At one point, for example, there was a local root exploit with the `eject` command on Solaris. Until there was

a patch available, you could have disabled that command using the following sequence:

```
disable:
   solaris::
      /usr/bin/eject inform=true syslog=true
```

This would not only disable the command, but it would inform the administrator and make a log entry using syslog.

Let's say that you want to remove `/etc/httpd/conf/httpd.conf`, if it exists, and create a symbolic link pointing to the master file `/usr/local/etc/httpd.conf`. The following command sequence can accomplish this task:

```
disable:
   /etc/httpd/conf/httpd.conf type=file define=link_httpd_conf
links:
   link_httpd_conf::
      /etc/httpd/conf/httpd.conf -> /usr/local/etc/httpd.conf
```

The `disable` section would only remove the file if it is a normal file (and not a link, for example). If the file is disabled, the `link_httpd_conf` class will be defined. Then, in the `links` section, a symbolic link will be created in its place.

It is important to remember that cfengine does not execute these sections in any predefined order. The `actionsequence` setting in the `control` section controls the order of execution. So, for this example, we need to make sure that the file is disabled before the symbolic link is created:

```
control:
   actionsequence = ( disable links )
```

You can also use the `disable` section to rotate log files. The following sequence rotates the web server access log if it is larger than five megabytes and keeps up to four files:

```
disable:
   /var/log/httpd/access_log size=>5mbytes rotate=4
```

This, and other method of rotating files, are further discussed in Chapter 9.

6.4.4.4 The editfiles Section

The `editfiles` section can be the most complex section in a configuration file. There are approximately one hundred possible commands that can be used in this section. These commands allow text files (and, in a few cases, binary files) to be checked and modified. Here is an example:

```
editfiles:
   { /etc/crontab
      AppendIfNoSuchLine "0 * * * * root /usr/local/sbin/cfexecd -F"
   }
```

This command adds the specified line of text to `/etc/crontab`. This makes sure that cron runs `cfexecd` every hour. You also might want to make sure that other hosts can access and use the printers on your printer servers as follows:

```
editfiles:
   PrintServer::
      { /etc/hosts.lpd
         AppendIfNoSuchLine "host1"
         AppendIfNoSuchLine "host2"
      }
```

In your environment, perhaps a standard port is used for another purpose. For example, you might want to rename port 23 to `myservice`. To do this, you could change its label in `/etc/services` on every host:

```
editfiles:
   { /etc/services
      ReplaceAll "^.*23/tcp.*$" With "myservice 23/tcp"
   }
```

If you are using inetd and want to disable the TELNET application, for example, you could comment out those lines in `/etc/inetd.conf`:

```
editfiles:
   { /etc/inetd.conf
      HashCommentLinesContaining "telnet"
      DefineClasses "modified_inetd_conf"
   }
processes:
   modified_inetd_conf::
      "inetd" signal=hup
```

Any line containing the string `telnet` will be commented out with the `#` character (if not already commented). If such a change is made, the inetd process is sent the `HUP` signal in the `processes` section.

6.4.4.5 The files Section

The `files` section can process files (and directories) and check for valid ownership and permissions. It can also watch for changing files. Here is a simple example:

```
files:
   /etc/passwd mode=644 owner=root group=root action=fixall checksum=md5
   /etc/shadow mode=600 owner=root group=root action=fixall
   /etc/group mode=644 owner=root group=root action=fixall

   web_servers::
      /var/www/html r=inf mode=a+r action=fixall
```

We accomplish several tasks with these entries. On every system, the ownership and permissions are checked (and fixed) on `/etc/passwd`, `/etc/shadow`, and `/etc/group`. The md5 checksum of `/etc/passwd` is also calculated and recorded.

On any system in the class `web_servers`, the permissions on `/var/www/html/` are checked. The directory is scanned recursively and all files and directories are made publicly readable. The execute bits on directories will also be set according to the read bits; so, since we requested files to be publicly readable, directories will also be publicly executable.

The `checksum` option requires a little more explanation. Since the file's checksum is stored, if the checksum changes at a later date, the administrator will be warned. In fact, the administrator will be warned every hour unless you configure cfengine to update the checksum database. In that case, the administrator will only be notified once, and then the database will be modified. You can enable this by adding the following command in the `control` section:

```
control:
   ChecksumUpdates = ( on )
```

Here are some other options you may want to use in this section:

`links`: Can be set to `traverse` to follow symbolic links pointing to directories. Alternatively, this can be set to `tidy` to remove any dead symbolic links (links that do not point to valid files/directories).

ignore: The `ignore` option can be specified multiple times. It requires a regular expression pattern or a simple string. Any file or directory matching this pattern is ignored. For instance, `ignore="^\."` would ignore all hidden files and directories.

include: If any `include` options are listed, any files must match one of these regular expressions to be processed.

exclude: Any file matching any of the `exclude` regular expressions will not be processed by `cfengine`.

define: If any changes are made to any file, this class will be defined. You can also list several classes, separated by colons.

elsedefine: If no changes are made to any files, this class will be defined. You can also list several classes, separated by colons.

syslog: When set to `on`, cfengine will log any changes to the system log.

inform: When set to `on`, cfengine will log any changes to the screen (or sent via email if so configured).

6.4.4.6 The links Section

With the `links` section, cfagent can create symbolic links.

```
links:
   /usr/tmp -> ../var/tmp
   /usr/local/bin +> /usr/local/lib/perl/bin
```

In this example, the first command creates (if it doesn't already exist) a symbolic link from `/usr/tmp` to `../var/tmp` (relative to `/usr/tmp`).

The second command creates one link in `/usr/local/bin/` pointing to each file in `/usr/local/lib/perl/bin/`. Using this technique, you could install applications in separate directories and create links to those binaries in the `/usr/local/bin/` directory.

There are plenty of possible options in the `links` section but, in practice, they are rarely used, so I will not cover them in this book. See the cfengine reference manual for more information.

6.4.4.7 The processes Section

System processes can be monitored and manipulated in the `processes` section. Here is an example from earlier in this chapter:

```
processes:
   "cfservd" restart "/var/cfengine/bin/cfservd"
   "cfexecd" restart "/var/cfengine/bin/cfexecd"
```

For the `processes` section, cfengine runs the `ps` command with either `-aux` or `-ef` switches (as is appropriate for the specific system). This output is cached and the first part of each command in the `processes` section is interpreted as a regular expression against this output. If there are no matches, the `restart` command is executed.

You can specify the following options when using the `restart` facility: `owner`, `group`, `chroot`, `chdir`, and/or `umask`. These affect the execution environment of the new process as started by the `restart` command.

You can also send a signal to a process:

```
processes:
   "httpd" signal=hup
```

This signal would be sent on every execution to any processes matching the regular expression `httpd`. It is also possible to specify limits on the number of processes that can match the regular expression. Let's say, for example, that you wanted to make sure there were not more than ten `httpd` processes running at any given time:

```
processes:
   "httpd" action=warn matches=<10
```

6.4.4.8 The shellcommands Section

For some custom and/or complex operations, you will need to execute one or more external scripts from cfengine. You can accomplish this with the `shellcommands` section. Here is an example:

```
shellcommands:
   all::
      "/usr/bin/rdate -s ntp1" timeout=30
   redhat.Hr02_Q1::
      "/usr/local/sbin/log_packages" background=true
```

On all systems, the `rdate` command is executed to synchronize the system's clock. This command is terminated by cfengine in 30 seconds if it has not completed. On systems running Red Hat Linux, when it is between 2:00AM and 2:15AM, a script runs to log the currently installed packages. This command is placed in the

background and cfengine does not wait for it to complete. This allows cfengine to perform other tasks or exit while the command is still running.

You can specify the following options to control the environment in which the command is executed: `owner`, `group`, `chroot`, `chdir`, and/or `umask`.

If these scripts want to access the list of currently defined classes, they can look in the `CFALLCLASSES` environment variable. Each active class will be listed, separated by colons.

Scripts, by default, are ignored when cfengine is performing a dry run (with `--dry-run` specified). You can override this by specifying `preview=true`. The script should, however, not make any changes when the class `opt_dry_run` is defined.

6.4.5 Using cfrun

The `cfrun` command allows you to execute `cfagent` on any number of systems on the network. It requires a configuration file in the current directory named `cfrun.hosts` (or a file specified with the `-f` option). The contents of the file should be as follows:

```
domain=mydomain.com
server.mydomain.com
client1.mydomain.com
client2.mydomain.com
```

Apart from the domain setting, this file is just a list of every host, including the configuration server. You can also have the output logged to a series of files (instead of being displayed to the screen) by adding these options to the top of the file:

```
outputdir=/tmp/cfrun_output
maxchild=10
```

This tells `cfrun` to fork up to ten processes and place the output for each host in a separate file in the specified directory. You can normally run `cfrun` without arguments. If you do want to specify arguments, the format is as follows:

```
cfrun CFRUN_OPTIONS HOSTS -- CFAGENT_OPTIONS -- CLASSES
```

`CFRUN_OPTIONS` is optional and can contain any number of options for the `cfrun` command. Next, you can specify an optional list of hostnames. If some hostnames are specified, only those hosts will be contacted. If no hosts are specified, every host in the `cfrun.hosts` file is contacted.

After the first -- are any options that you want to pass to the actual `cfagent` command run on each remote system. After the second -- is an optional list of classes. If some classes are specified, only hosts that match one of these classes will actually execute `cfagent` (although each host is contacted because each host has to decide if it matches one of the classes).