A Site Configuration Engine

Mark Burgess Oslo College and Oslo University

ABSTRACT: Cfengine is a language-based system administration tool in which system maintenance tasks are automated and the configuration of all networked hosts are defined in a central file. Host configuration may be tested and repaired any number of times without the need for human intervention. Cfengine uses a decision-making process based on class membership and is therefore optimized for dealing with large numbers of related hosts as well as individually pin-pointed systems.

1. Introduction

The proliferation of TCP/IP networks, combined with the increased availability of cheap UNIX-like solutions, continues to make machine-parks grow at a rate which keeps system administrators on their toes. This presents a practical difficulty to administrators: how does one keep track of hundreds or perhaps thousands of systems and be sure that they are configured according to the network standard? In spite of the efforts of standardizing organizations, the various operating system alternatives from major software developers are all substantially different from a system administrator's perspective and, on a large heterogeneous network, one is forced to undergo an often tiresome process of adapting each type of system in order to make the alternatives cooperate harmoniously. Traditionally, such fixes have been made by hand or with the help of shell scripts—a procedure which becomes increasingly cumbersome and haphazard as the size of a network expands beyond a handful of systems. A viable tool for efficiently systemizing the administration of such a network has been lacking for some time.

The purpose of GNU cfengine [Burgess 1993, Burgess 1994, Burgess 1995] is to provide a high level, language-based interface for the task of system administration. Using cfengine, administrators can create a single file which defines the configuration of all hosts on an arbitrarily large network. Changes made in this single file can cause system-wide changes to take place, or can pin-point actions to be taken on a single host. The configuration language hides the differences between different operating systems and automates frequently performed tasks, thereby creating a very high level description. This can be used to both document and enforce the characteristics, interrelationships, and dependencies of all hosts from a single, easily readable file. A cfengine configuration program can be used to automatically set up a new host from scratch, making all the changes necessary to blend it into the local network; it can also be run an unrestricted number of times to check or maintain the state of that configuration. By defining system configuration in a central file, accidents which destroy the changes made on a special host are no longer a problem, since a single run of the systemwide program will restore the configuration to the defined standard.

The functionality of cfengine can be summarized by the following list:

- Testing and configuration of the network interface
- Simple automated text file editing
- Symbolic link management
- Testing and setting the permissions and ownership of files
- Systematic deletion of garbage files
- Systematic automated mounting of NFS filesystems
- Other sanity checks

This article is a short conceptual presentation of cfengine. Tutorials and more information can be obtained from the distributed documentation [Burgess 1995].

2. *Why a New Language?*

Cfengine's main contribution to system administration is to provide relevant tools for a limited number of frequently-used-operations. Cfengine supplements the functionality of peer languages, such as Perl and lower level scripting languages, by providing free checks which are built in to the engine itself. This frees programs from the clutter of irrelevant checking code and admits a more conceptually user-friendly interface. For example, the command to create a symbolic link in cfengine is

```
file1 -> file2
```

The corresponding command in shell is

```
ln -s file2 file1
```

The functionality which cfengine adds here is the following algorithm, which is executed for every single link defined in a cfengine program:

- Does link exist? If not, create it.
- Is the name a plain file or directory, not a link? If so signal a warning.
- If link exists, does it point to the location specified?
- If yes, do nothing, say nothing.
- If not, signal a warning.

The above algorithm is designed in such a way that it can be run an unlimited number of times without generating spurious and uninteresting output. In shell, the execution of the command `ln -s` twice results in

```
borg% ln -s  cv.tex bla
borg% ln -s cv.tex bla
ln: bla: File exists
```

an irrelevant and unhelpful error message. In cfengine, the execution of the link script n-times results in no output unless verbose mode is selected. Additional spices exist, such as the ability to link all of the children in a particular directory to corresponding files in another. Again, extensive checking both of new files and previously existing files is made.

In this example, cfengine has not provided anything which could not be reproduced in a shell script—what it has done is to simplify the code required to perform the appropriate actions considerably, by hiding the irrelevant details in the language definition. This is the function of high level languages. Similar features are true of the other operations performed by cfengine. Some of these will be mentioned in the remainder of the text.

3. Classes

One of the aims of cfengine is to make configuration programs as transparent as possible. A key design feature which makes this possible is the introduction of a class-based decision structure. A system-wide configuration program must make a considerable number of decisions in order to match statements to hosts. In a traditional scripting language this would mean coding a large number of `if...then...else` statements, perhaps nested many times. Since a test is required not only to determine which hosts a particular command applies to, but also to determine whether or not the present state of configuration is correct, the number of tests very easily accounts for the bulk of coding in any program. To avoid this scenario, cfengine uses a procedure of whittling away irrelevant statements by classifying them according to certain properties of the host executing the program.

A class-based decision structure is possible because a cfengine configuration program is run by every host on the network individually. Each host knows its own name, the type of operating system it is running and can determine whether it belongs to certain groups or not. Each host which runs a cfengine program therefore builds up a list of its own attributes (called classes). A class may, in fact, consist of the following:

- The hostname of a machine.
- The operating system and architecture of the host.
- A user-defined group to which the host belongs.
- A day of the week.
- The logical AND of any of the above.

Given that a host knows its own class attributes, it can now pick out what it needs from a list of commands provided the commands are also labelled with the classes to which they apply. A command is only executed if a given host is in the same class as the command it finds in the configuration program—a host can pick out only the commands which it knows apply to itself and ignore the others. There is no need for formal decision structures, it is enough to label each statement with classes. At the simplest level, one has commands belonging only to a single class, say the operating system type of the hosts:

```
ultrix::
```
statements

Here the statements which follow the class `ultrix` are executed only if the host is an Ultrix system. To combine classes, signifying multiple membership dots are used:

```
ultrix.Monday.mygroup::
```
statements

In this example, the statements which follow are only executed if the host is of type Ultrix, the day is Monday and the host is a member of the user-defined group `mygroup`.

User-defined classes can be defined and undefined on the command line and in the action sequence in order to switch certain statements on and off for special purposes. This makes it easy to isolate parts of a global configuration for partial execution. It is, for example, useful to mark very time-consuming operations with a class `heavy` which can then be undefined in order to execute a quick version of the program.

4. Syntax

The syntax employed by cfengine resembles in some ways a Makefile (see, for example, [Oram & Talbott, 1991]), where instead of targets one has classes. Each

cfengine program is a free format file composed of a number of sections. Each section deals with a particular task, such as symbolic links or file editing. Each section defines actions for various classes (see figure 1).

```
#
# Cfengine program
#
groups:
myclass = ( host1 host2 host3 +@NIS-netgroup )

control:
actionsequence = ( links files )
links:
class::
/tmp/x  -> /usr/tmp/x

class1.class2.class3::
/bin/tcsh -> /usr/local/bin/tcsh

files:
myclass::
/usr/local owner=root mode=o-w action=fixall
```

Figure 1. The form of a simple cfengine program. The format is free, use of space is arbitrary.

Class membership of statements is, in fact, optional: if no special membership is specified, a statement is assumed to belong to all classes and is executed on the host running the program.

Figure 1 illustrates, with a trivial example, the basic points of syntax in a cfengine program. Each program is a free format textfile containing symbol declarations and actions to be performed. More formally, it is a list composed of the elements of the form

```
section:
    class::
        statements
```

Omitting a *class* specifier is equivalent to using a wildcard class `any::` which means that the following statements are to be executed on all hosts. Statements have a syntax which depends on the section of the program: some of these will define symbolic links, others specify editing actions for files, etc.

The meaning of the example program in figure 1 is the following. The first few lines are comments and are identified by lines beginning with the hash symbol `#`. The `groups` section of a program defines a new class called `myclass`. This class has as its members host1, host2, host3, and all of the hosts in the netgroup NIS-netgroup. If the host which executes the cfengine program is one of those hosts, it inherits the class `myclass` and statements which also belong to `myclass` can then be executed.

In contrast to a makefile, the dependencies in a cfengine program are not files which must exist but *host attributes* which must be present. In a makefile, actions are performed if the target does not exist; in a cfengine program, actions are performed if the classes *do* match the present state of the system. In other words, a cfengine program is not an instruction of how to build a system, but a statement of what many different classes of system should look like. Some users have focused on the similarities with make and have suggested that the make program combined with shell scripts would do the same job. While it is certainly true that any problem can be solved in a variety of ways, the criticism is somewhat misguided since the real gains in using cfengine are that one avoids having to write long and complicated scripts employing repetitive checking procedures. Cfengine is a classic meta-language: it eliminates the need for tiresome repetitive coding by absorbing frequently used code back into the language.

The control part of a cfengine program is used to set certain internal variables and to define macros. The most important system variable is a list called the `actionsequence`. Without an action sequence, a cfengine program does nothing. It is a way of switching on and off certain statements. For example, if one adds the item `links` to the action sequence, cfengine will process all of the link commands which belong to classes the current host belongs to. The action sequence determines the order and number of times in which these bulk actions are carried out (the actual ordering of the declarations in a cfengine program is irrelevant and should be used to achieve conceptual clarity rather than to indicate the sequence of events). If this bulk handling of commands is too coarse, finer control is achieved by using the notation

```
actionsequence =
(
links.class1.class2
...
```

```
links.class3
)
```

which means: execute links commands—but, on the first pass, define the additional classes *class1* and *class2* for the duration of this pass only; on the second pass define the additional symbol *class3* for the duration of the pass. The result is that, in the first case, only links labelled by *class1* and *class2* will be executed and in the second case only links labelled with *class3* will be executed. Classes defined in the action sequence have no lasting effect. They are *local* to a given action and are only used to achieve a finer control over the sequence of execution. They are attributes of the current task rather than of the host.

The keywords or actions in the actionsequence are internally defined and are taken from the following list, which is printed incidentally in the order in which the actions might typically be called:

```
mountall            # mount filesystems in fstab
mountinfo           # scan mounted filesystems
checktimezone       # check timezone
netconfig           # check net interface config
resolve             # check resolver setup
unmount             # unmount any filesystems
shellcommands       # execute shell commands
editfiles           # edit files
addmounts           # add new filesystems to system
directories         # make any directories
links               # check and maintain links
mailcheck           # check mailserver
required            # check required filesystems
tidy                # tidy files
disable             # disable files
files               # check file permissions
```

For a full explanation of these functions, the reader is referred to the cfengine documentation.

5. *Functions*

In this section, a cursory overview of the functionality of cfengine is presented.

5.1. Network

The configuration of the ethernet interface is one of the prerequisites for getting a host up and running. It includes informing the ethernet interface of the subnet-mask, broadcast address, and default route of the host. In addition, the Domain Name Service has to be configured. These tasks are handled by cfengine at a high level. It is sufficient to define:

- the value of the internal variable `netmask`,
- the bit-convention for determining the broadcast address (either all ones or all zeros),
- the default route for packets (normally the address of the local gateway),
- the system domain name,
- an ordered list of nameservers.

These can naturally be specified either once for all hosts or individually by special classes, depending on the physical organization of the net.

If the appropriate directives are added to the action sequence, cfengine uses this information to check the present state of the ethernet device and, if necessary, configure it to the standard defined in the configuration program. The default route is added to the static routing table if necessary. Cfengine then loads the file `resolv.conf`, and ensures that the DNS domain name is correct and that the correct nameservers are present with the defined priority.

5.2. File Editing

One of the characteristics of BSD/System V systems is that they are configured primarily by human-readable textfiles. This makes it easy for humans to configure the system and it also simplifies the automation of the procedure. Most configuration files are line-based text files, a fact which explains the popularity of, for example, the Perl programming language [Wall & Schwarz 1990]. Cfengine does not attempt to compete with Perl or its peers. Its internal editing functions operate at a higher level and are designed for transparency rather than flexibility. Fortunately most editing operations involve appending a few lines to a file, commenting out certain lines or deleting lines. Files are edited with commands from the following list:

```
DeleteLinesStarting "text..."
DeleteLinesContaining "text..."
```

```
AppendIfNoSuchLine "text..."
PrependIfNoSuchLine "text..."
WarnIfNoSuchLine "text..."
WarnIfLineMatching "text..."
WarnIfLineStarting "text..."
WarnIfLineContaining "text..."
WarnIfNoLineStarting "text..."
WarnIfNoLineContaining "text..."
HashCommentLinesContaining "text..."
HashCommentLinesStarting "text..."
HashCommentLinesMatching "text..."
SlashCommentLinesContaining "text..."
SlashCommentLinesStarting "text..."
SlashCommentLinesMatching "text..."
PercentCommentLinesContaining "text..."
PercentCommentLinesStarting "text..."
PercentCommentLinesMatching "text..."
```

Commands containing the word "comment" are used to *comment out* certain lines from a textfile—i.e. render a line impotent without actually deleting it. Three types of comment are supported: shell-style (hash) #, % as used in TeX and on AIX systems, and C++-style //.

An example of the use of this might be the following. Each new GNU/Linux installation contains a line in the start-up scripts which deletes the contents of the "message of the day" file each time the system boots. On a system which boots often this would be irritating. This line could be commented out for every GNU/Linux system on the network with a simple command:

```
editfiles:

   linux::

      { /etc/rc.d/rc.S

      HashCommentLinesContaining "motd"
      }
```

Other applications for these editing commands include monitoring and controlling root-access to hosts by editing files such as `.rhosts` and setting up standard environment variables in global shell resource files—for example, to set the timezone.

Files are loaded into cfengine and edited in memory. They are only saved again if modifications to the file are carried out, in which case the old file is preserved by adding a suffix to the filename. When files are edited, cfengine generates a warning for the administrator's inspection so that the reason for the change can be investigated.

The behavior of cfengine should not be confused with that of `sed` or `perl`. Again, it is true that nothing really new is introduced, but that a considerable saving of user-programming is involved—moreover a common interface is used, taking full advantage of the class selectors. Some functionality is reproduced for convenience, but the specific functions have been chosen on the basis of (i) their readability and (ii) the fact that they are frequently-required-functions. A typical file editing session involves the following points:

- Load file into memory.
- Is the size of the file within sensible user-definable limits? If not, file could be binary, refuse to edit.
- Check each editing command and count the number of edits made.
- If number of edits is greater than zero, rename the old file and save the edited version in its place. Inform about the edit.
- If no edits are made, do nothing, say nothing.

Equivalent one-line sed operations involve editing the same file perhaps many times to achieve the same results—without the safety checks additional.

5.3. Mount Model

Cfengine regards NFS filesystems as resources. Resources, like actions, also belong to classes and are mounted on the basis of class decisions. Cfengine automates the mount procedure as far as possible; administrators have only to specify a number of servers for a class of hosts and cfengine will edit the appropriate filesystem tables and attempt to mount the resources automatically.

Cfengine distinguishes between two types of mountable resources which it refers to as *binary filesystems* and *home filesystems*. Binary filesystems contain architecture-specific data—i.e. compiled software which only applies to the operating system under which it was compiled. Home filesystems contain users' login areas and can be mounted meaningfully on any type of host. The way information is structured in cfengine programs makes mounting of binary and home resources quite transparent. For each class of hosts one defines a number of binary servers and a number of home servers. Cfengine mounts automatically all the

declared resources from all a host's servers by referring to a list which contains every filesytem resource available on the network. Network resources are defined like this:

```
mountables:

     server:/site/server/home1
     server:/site/server/home2
     server2:/site/serv2/local
```

The name of the server (preceding the colon) and the remote directory name (following the colon) are declared in this list so that cfengine can search for resources of different types. Employing a user-definable pattern, cfengine can distinguish between home and binary resources, and mount the appropriate resources on directories with the same names as the source filesystems. Note that the key to the success of this model is that remote filesystems are mounted on directories with the same name on the local host. This is not a restriction provided one uses a rational naming scheme and any anomalies can be handled by the *miscellaneous mount* command (which is more awkward syntactically but lifts the naming restriction).

To make the scheme work then, it is necessary to introduce a strict naming convention for filesystem mount-points.[1] While this is user-configurable, the recommended convention is to mount all filesystems according to a three component directory name:

/site/hostname/file-system-name

in which the site name is the name of your local department or section (separate subnet), the hostname is the name of the host which is the server for the filesystem and the final link is the name of the directory itself. Strict adherence to this system means that no two filesystems will ever collide. Symbolic links can then be used to make cosmetic changes to the system, for example to create an alias from `server2:/site/serv2/local` to `/usr/local`.

The issue of editing the exports files on the servers is not addressed directly by cfengine since there is no unique way of handling this issue. If necessary it could be dealt with using the editfiles facility. In practice it easier to deal with exports by hand—if only for security reasons.

1. This naming convention was first suggested to me by Knut Borge of USIT, University of Oslo.

The model cfengine uses for mounting filesystems around the network is simple and effective. The amount of writing required to add a large number of filesystems to either a single host or a class of hosts is simply equal to the number of servers on which the resources reside.

Although most filesystems fall into the categories binary and home, some—like information databases and sharable resources—do not. These remaining resources can be dealt with using a miscellaneous mount command which makes no reference to a special model. A small amount of extra writing is required in this case. For example:

```
miscmounts:

   myhost::

      otherhost:/site/otherhost/info /library/database rw
```

Cfengine hard-mounts filesystems by default. In contrast to the NFS automounter [Sun Microsystems] the filesystems are mounted by editing the filesystem table so that all filesystems are available from boot time. Hence the functionality does not compete with the automounter but augments it.

5.4. Files and Links

File and link management takes several forms. Actions are divided into three categories called `files`, `tidy`, and `links`. The first of these is used to check the existence, ownership, and permissions of files. The second concerns the systematic deletion of garbage files. The third is a link manager which tests, makes, and destroys links.

The monitoring of file access bits and ownership can be set up for individual files and for directory trees, with controlled recursion. Files which do not meet the specified criteria can be fixed—i.e. automatically set to the correct permissions—or can simply be brought to the attention of the system administrator by a warning. The syntax of such a command is as follows:

```
files:
     class::
        /path mode=mode owner=owner group=group
                    recurse=no-of-levels action=action
```

The directory or file name is the point at which cfengine begins looking for files. From this point the search for files proceeds recursively into subdirectories with a maximum limit set by the `recurse` directive, and various options for

dealing with symbolic links and device boundaries. The mode-string defines the allowed filemode (by analogy with `chmod`) and the owner and group may specify lists of acceptable user-ids and group-ids. The action taken in response to a file which does not meet acceptable criteria is specified in the action directive. It includes warning about or directly fixing all files, or plain files or directories only. Safe defaults exist for these directives so that in practice they may be treated as options.

For example,

```
files:

  any::
       /usr/*/bin mode=a+rx,o-w own=root r=inf act=fixall
```

which (in abbreviated form) would check recursively all files and directories starting from directories matching the wildcard (e.g. `/usr/local/bin`, `/usr/ucb/bin`). By default, `fixall` causes the permissions and ownership of the files to be fixed without further warning.

The creation of symbolic links is illustrated in figure 1 and the checking algorithm was discussed in section 2. In addition to the creation of single links, one may also specify the creation of multiple links with a single command. The command

```
links:

   binaryhost::

      /local/elm/bin +> /local/bin
```

links all of the files in `/local/elm/bin` to corresponding files in `/local/bin`. This provides, amongst other things, one simple way of installing software packages in regular bin directories without controlling users' PATH variable. A further facility makes use of cfengine's knowledge of available (mounted) binary resources to search for matches to specific links. Readers are referred to the full documentation concerning this feature.

The need to tidy junk files has become increasingly evident during the history of cfengine. Files build up quickly in areas like `/tmp/`, `/var/tmp`. Many users use these areas for receiving large ftp-files so that their disk usage will not be noticed! To give another example, just in the last few months the arrival of netscape [Netscape Communication Corp. 1995] World Wide Web client, with its caching facilities, has flooded harddisks at Oslo with hundreds of megabytes of

WWW files. In addition the regular appearance of `core` files[2] and compilation by-products (`.o` files, `.log` files, etc.) fills disks with large files which many users do not understand. The problem is easily remedied by a few lines in the cfengine configuration. Files can be deleted if they have not been accessed for n-days. Recursive searches are both possible and highly practical here. In the following example:

```
tidy:

   AllHomeServers::

      home                 pattern=core       r=inf age=0
      home/.wastebacket    pattern=*          r=inf age=14
      home/.netscape-cache pattern=cache????* r=inf age=2
      home/.MCOM-cache     pattern=cache????* r=inf age=2
```

all hosts in the group `AllHomeServers` are instructed to iterate over all users' home directories (using the wildcard `home`) and look for files matching special patterns. Cfengine tests the *access time* of files and deletes only files older than the specified limits. Hence all core files, in this example, are deleted immediately, whereas files in the subdirectory `.wastebasket` are deleted only after they have lain there untouched for 14 days, and so on.

5.5. Calling Scripts

Above all, the aim of cfengine is to present a simple interface to system administrators. The actions which are built into the engine are aimed at solving the most pressing problems, not at solving every problem. In many cases administrators will still need to write scripts to carry out more specific tasks. These scripts can still be profitably run from cfengine. Variables and macros defined in cfengine can be passed to scripts so that scripts can make maximal advantage of the class based decisions. Also note that, since the days of the week are also classes in cfengine, it is straightforward to run weekly scripts from the cfengine environment (assuming that the configuration program is executed daily). An obvious use for this is to update databases, like the fast-find database one day of the week, or to run quota checks on disks. A disk backup script is included in the distribution.

2. On some systems, core dumps cannot be switched off!

```
shellcommands:

   myhost.Sunday::

      "/usr/bin/find/updatedb"
```

6. *How Cfengine Is Run*

Cfengine was designed to be run as a batch job, ideally at night when system disk load is low. Because its policy is to check and then correct, it can also be run manually any number of times without ill effects. Cfengine runs silently by default, producing a message only if something is wrong. It is therefore convenient to have error messages mailed to the system administrator. This is accomplished by running cfengine from a wrapper script which reports the name of the host and forwards the text from cfengine. Suitable wrapper scripts are included with the cfengine distribution.

Since cfengine only acts when action needs to be taken, a cfengine program can be run any number of times without harmful side effects. A typical scenario is the following. On the arrival of a new machine, a single NFS directory is then mounted by hand to gain access to a compiled version of cfengine and the global configuration file. Cfengine is run and the machine is instantly configured—all symbolic links, NFS filesystems, and textfiles are in place. The host is now installed. This should be sufficient. A reboot of the host should now have no effect on the configuration. Cfengine can itself be programmed to add itself to the cron file so that it is run each night so as to monitor the host on a regular basis.

The global cfengine program can also profitably be called up in the system boot scripts `/etc/rc.local` or its equivalent, perhaps with certain actions excluded to save time. It can be used to set the netmask, broadcast address, and default route as well as checking the ordering of nameservers in `/etc/resolv.conf` each time the system boots.

7. *Security*

Cfengine has built-in features which are designed for system security. The ability to monitor file permissions and ownership is the first step. A common problem is that files obtained by an ftp session get transferred with a user-id which belongs to a completely random user on the local system. This can either cause access

problems or compromise the security of the files. A busy administrator could easily overlook this or simply forget to change the ownership of the files. A routine check of all files would discover this fact very quickly.

A by-product of the file checking is that cfengine maintains a list of all known setuid-root programs and setgid-root programs which it finds in the course of checking the system. When a new setuid-root program appears on the system, a warning is always issued so that any potentially dangerous software is brought to the administrator's attention. In most cases it will be the administrator who has installed the software, but on other occaisions this could help to reveal surreptitiously installed programs.

Using cfengine as a scripting language is also made safer. If a cfengine script is made setuid-root (on a system which allows you to do this), it is still possible to restrict the users who can run that script as a secondary check. For example:

```
access = ( mark root )
```

An access control list defines the usernames who may run a program. This makes it easy to write a program which can be run by others to fix a particular problem on the system. Responsibility can thus be disseminated quite safely to system helpers.

Cfengine does not have to be run setuid-root, nor do any of its features demand the availability of this feature. However, on systems which do support this option, it is presumed that this will be a helpful additional feature. Caution should always be exercised when opening privileged access to non-privileged users.

8. *Scripting Language*

Although the focus of attention has always been the construction of systemwide configuration files, cfengine can also be used to write smaller scripts. For example, the following script provides a useful way for users to manage their own files, opening files for collaboration with other users and closing others which are private.

```
#!/local/gnu/bin/cfengine -f
#
# Open my shared directory for others in my group
#

control:
```

```
  actionsequence = ( files )

files:

  $(HOME)          mode=a+rx r=0 action=fixdirs
  $(HOME)/share    mode=ug+rw,o-rwx r=inf group=share act=fixall
  $(HOME)/private mode=0600 r=inf action=fixall
```

The first line ensures that the user running the script has a home directory which is open to other users. The second line opens the subdirectory `share` to the group `share` and tells cfengine to fix the files recursively. Note that, in recursive searches, cfengine will automatically set the `x` flag on directories if the corresponding `r` flag is defined.

9. *Experience*

Cfengine has been on test, in prototype form, for three years during its development. In addition the recent GNU release is now in use at least twenty sites around the world. The number of features has grown in accordance with experience in using it and for its GNU release the syntax has been altered radically from earlier versions. New features are incorporated as feedback is received through the official mail point `bug-cfengine@prep.ai.mit.edu`

The philosophy employed in writing the configuration scripts has been to define as many general rules as possible. Special exceptions are to be avoided since they increase the size of the configuration and make programs harder to understand. This might give the impression of a loss of flexibilty, but systematic administration procedures on a large scale are by necessity simple-minded and general. More difficult, specific issues can be dealt with locally, using local scripts (written in cfengine or some other utility) and controlled by individuals who are closer to the individual host concerned. In most cases, special configuration requirements are a result of specially licensed software which runs only on a single host, or perhaps a small cluster of hosts. These can nearly always be integrated into the global configuration by using symbolic links. Cfengine has two powerful features for building and managing a large number of symbolic links automatically. Indeed, experience shows that cfengine would be a useful tool if the only thing it did was to manage symbolic links. The use and maintainance of links (whose names can be based on systemwide variables) opens up a new way of making easily understandable and *maintainable* patches to systems.

Certain habitual practices must naturally be relearned in order to make effective use of cfengine: administrators, used to configuring systems by hand, have to discipline themselves to make changes only in the configuration file and then run cfengine to make a change. Initially this introduces an extra step, and therefore a certain amount of resistance, but on networks supporting hundreds of hosts this minor overhead is worth the potential rewards.

10. Example Program

Here is a more substantial example program to illustrate the uses for cfengine. Follow the comments for the details. It is difficult to represent all of the useful features here; hopefully there is enough in this example to whet the appetite for more.

```
#########################################################
#
#  CFENGINE CONFIGURATION FOR site = iu.hioslo.no
#
#########################################################


groups:

   science = ( nexus ferengi regula borg dax lore axis )
   diskless = ( regula ferengi lore )

   AllHomeServers   = ( nexus )
   AllBinaryServers = ( nexus borg )

   OIH_servers = ( nexus borg )
   OIH_clients = ( ferengi regula dax lore )

   XTerminalServer = ( nexus )
   WWWServers = ( nexus )
   FTPserver = ( nexus )

   LPD_clients = ( ferengi regula borg dax lore axis )


#########################################################
```

```
control:

   access    = ( root )     # only root gets to start this

   site      = ( iu )
   domain    = ( iu.hioslo.no )
   sysadm    = ( sysadm@iu.hioslo.no ) # errors to ..

   netmask   = ( 255.255.255.0 )
   timezone  = ( MET )
   nfstype   = ( nfs )

   sensiblesize  = ( 1000 )  # missing filesystem if total bytes
                             # in fs less than 1000 (arbitrary)
   sensiblecount = ( 2 )     # missing filesystem if total files
                             # in fs less than 2 (arbitrary)
   editfilesize  = ( 6000 )  # Safety: don't edit files bigger than
                             # 6000 bytes - could be a mistake!

   actionsequence =          # Checking order...
      (
      mountall
      mountinfo
      checktimezone
      netconfig
      resolve
      unmount
      shellcommands
      editfiles
      addmounts
      directories
      links
      mailcheck
      mountall
      required
      tidy
      disable
      files
      )

   mountpattern = ( /$(site)/$(host) )

     # user dirs are u1, u2, etc

   homepattern = ( u? )
```

```
   addclasses = ( exclude )

 #
 # Macros & constants
 #

   main_server = ( nexus )
   gnu_path = ( /local/bin/gnu )
   ftp = ( /local/ftp )

########################################################

 # Nexus is the only host holding users' home dirs, so we
 # have to mount these on all systems listed in science

homeservers:

   science:: nexus

 # nexus and borg hold the binaries for /local for their
 # respective OS types...  so any machines of these types
 # in science should mount all non-home dirs from the
 # list of mountables. In this case there is only
 # .../local to mount, but there could be any number
 # handled by this one command.

binservers:

   science.solaris::     nexus
   science.linux::       borg

  # The mail intray is on nexus and (on nexus) is called
  # /var/mail. This will be mounted where the local OS
  # expects to find it e.g. /usr/spool/mail on BSD.

mailserver:

   any::
          nexus:/var/mail

  # This is a list of all mountable partitions
  # available by NFS. (Used by binservers/homesevers)

mountables:

   any::
```

```
        nexus:/iu/nexus/u1
        nexus:/iu/nexus/u2
        nexus:/iu/nexus/local
        borg:/iu/linux/local

  # An exception to a general rule - here it proves
  # convenient to mount a solaris binary fs onto a
  # linux machine because it contains some config
  # files which are useful.

miscmounts:

   borg::   nexus:/iu/nexus/local /iu/nexus/local ro


########################################################

import:

  # Some rules can be made so general that they can be
  # collected into a separate file to make this file
  # less cluttered.

   any::      cf.global_classes
   linux::    cf.linux_classes

########################################################

broadcast:

 # All our networks use the newer 'ones' convention
 # for broadcasting, but some still use zeroes.

  ones

 # Set a default route to the local gateway for all
 # hosts

defaultroute:

  oih-gw

########################################################

resolve:
```

```
  # Our nameservers (applies to all hosts)

      128.39.89.10
      158.36.85.10
      129.241.1.99

##########################################################

links:

  # Everyone needs a local dir. $(binserver) expands to
  # hostname if that dir exists -- if not it expands to the

      /local -> /$(site)/$(binserver)/local

  # Make sure we dispose of silly sendmail and replace it
  # with Berkeley V8 in /local/mail

   solaris::

      /usr/lib/sendmail     ->! /local/mail/bin/sendmail
      /etc/mail/sendmail.cf ->! /local/mail/etc/sendmail.cf

  # Link some packages into /local/bin so we don't have
  # to have a 10 mile long PATH variable...

   nexus::

     /local/bin       +> /local/perl/bin
     /local/bin       +> /local/elm/bin

##########################################################

tidy:

  # List some files we want *deleted* once and for all...
  # The age refers to the access time of the files...
  # First tidy the users' home dirs, then the tmp areas.

   AllHomeServers.exclude::

       home                pat=core      r=inf age=0
       home                pat=a.out     r=inf age=2
       home                p=*%          r=inf age=2
       home                p=*~          r=inf age=2
```

```
      home                  p=#*           r=inf age=1
      home                  p=*.dvi        r=inf age=14
      home/.wastebacket     p=*            r=inf age=14
      home/.netscape-cache  p=cache????*   r=inf age=2
      home/.MCOM-cache      p=cache????*   r=inf age=2

   any::

      /tmp/                 pat=*          r=inf A=1
      /var/tmp              pat=*          r=inf A=1
      /                     pat=core       r=1   A=0

#############################################################

files:

   # All the local binaries should be owned by root
   # and nothing should be writable to the world!

   AllBinaryServers.exclude::

      /local mode=-0002 r=inf owner=root group=0,1,2,3,4,5,6

   # Make sure that none of the users' files are unwittingly
   # writable by others and delete any links which point
   # nowhere and confuse everyone.
   # Note ''ignore'' exception for www directory below, since
   # some users want to allow user nobody to be able to edit a
   # guestbook file...

   AllHomeServers.exclude::

      home m=o-w R=inf act=fixall links=tidy

   # Make sure the local ftp dirs have the right
   # permissions...

   FTPserver.solaris::

      $(ftp)/pub  mode=755 o=ftp g=ftp r=inf act=fixall
      $(ftp)/Obin mode=111 o=root g=other act=fixall
      $(ftp)/etc  mode=111 o=root g=other act=fixdirs
      $(ftp)/usr/bin/ls mode=111 o=root g=other act=fixall
      $(ftp)/dev mode=555 o=root g=other act=fixall
      $(ftp)/usr mode=555 o=root g=other act=fixdirs
```

```
#########################################################

directories:

   solaris::

      /usr/lib/X11/nls  # for httpd

   borg::

      /local/tmp  mode=1777 o=root g=0

#########################################################

ignore:

   # Don't enter these directories in recursive descents

   any::

      .X11
      !*
      /local/lib/gnu/emacs/lock/
      /local/tmp
      /local/ftp
      /local/bin/top
      /local/lib/tex/fonts
      /local/etc
      /local/www
      /local/httpd_1.4/conf
      /local/mutils/etc/finger.log

  # For users' home dirs, so ''nobody'' can edit the guestbook

      www

#########################################################

required:

   # All hosts should have access to the /local dir. Warn if
   # they don't, or it looks funny (sensiblesize, sensiblecount)

      /${faculty}/${binserver}/local

#########################################################
```

```
editfiles:

  # Some basic files to edit.

   solaris::

      { /etc/netmasks

      DeleteLinesContaining "255.255.254.0"
      AppendIfNoSuchLine "128.39  255.255.255.0"
      }

   # cfengine installs itself as a cron job.

      { /var/spool/cron/crontabs/root

      AppendIfNoSuchLine "0 0 * * * \
      /local/gnu/lib/cfengine/bin/cfwrap \
      /local/gnu/lib/cfengine/bin/cfdaily"
      }

   nexus::

      { /etc/services

      WarnIfNoLineContaining "http"
      WarnIfNoLineContaining "pop"
      WarnIfNoLineContaining "bootpc          68/udp"
      WarnIfNoLineContaining "bootp           67/udp"
      }

      { /etc/inetd.conf

      AppendIfNoSuchLine "bootp dgram udp wait root \
      /local/bin/bootpd bootpd -i -d"
      }

   any::

      { /etc/shells

      AppendIfNoSuchLine "/local/bin/tcsh"
      }

#########################################################
```

```
shellcommands:

  # Update the find/locate databases and the
  # manual key on Sundays...

   AllBinaryServers.solaris.exclude::

      "/local/gnu/lib/locate/updatedb"

   AllBinaryServers.sun4.Saturday.exclude.Sunday::

      "/usr/bin/catman -w -M /local/man"
      "/usr/bin/catman -w -M /local/X11R5/man"
      "/usr/bin/catman -w -M /usr/man"
      "/usr/bin/catman -w -M /local/gnu/man"


##########################################################

disable:

 # Good to disable log files periodically so they don't
 # grow too big!

 WWWServers.Sunday::

   /local/httpd_1.4/logs/access_log
   /local/httpd_1.4/logs/agent_log
   /local/httpd_1.4/logs/error_log
   /local/httpd_1.4/logs/referer_log

  # Disable sendmail if it's a file. If it's the link
  # we made further up, leave it!
  # Also delete standard .login file which tcsh can't
  # understand.

 solaris::

    /usr/lib/sendmail type=file
    /etc/.login type=file
```

11. Summary

Cfengine is a language based interface for automating key areas of system administration on potentially large TCP/IP networks. The configuration of all hosts on a local area network may be steered from a single, central program, whose primary aim is to be as simple as possible to understand. Cfengine enhances the functionality of shell programs and provides an integrated environment for system configuration which avoids excessive CPU usage (pipes) and minimizes disk accesses. The full functionality of the engine has not been discussed in this article: readers are referred to the GNU package itself for comprehensive documentation and examples.

Future enhancements include the further development of the text editing facilities and the possibility of interfacing to companion tools for process monitoring in real time. Cfengine could also be enhanced by the introduction of a daemon which ensured that it was run (albeit silently) on every host. Ideally, cfengine configuration files would be available in a distributed database such as NIS.

Cfengine can be obtained by anonymous ftp from ftp.iu.hioslo.no and from any GNU site. A list of GNU sites can be obtained by connecting to `prep.ai.mit.edu` by anonymous ftp in file `/pub/gnu/GNUinfo/FTP`. The current version at the time of writing is 1.2.10 and it runs on SunOS/Solaris, HP UX, ULTRIX, IRIX, OSF 1, LINUX, and AIX.

I am grateful to Richard Stallman, Ola Borrebæk, and Morten Hanshaugen for their constructive criticisms.

References

1. M. Burgess, Cfengine, University of Oslo report, 1993.
2. Cfengine was first presented publicly at the CERN HEPIX meeting, France, M. Burgess, October 1994.
3. M. Burgess, Cfengine documentation, Free Software Foundation, 1995.
4. The Netscape program, Netscape Communications Corporation, `http:/home.netscape.com.`, 1994.
5. A. Oram and S. Talbott, *Managing projects with make*, O'Reilly & Assoc., 1991.
6. Sun Microsystems, The NFS automounter, SunOS/Solaris manual pages.
7. L. Wall and R. Schwarz, *Programming perl*, O'Reilly & Assoc., 1990.