cfengine reference

Edition 2.2.1 for version 2.2.1

Mark Burgess
Faculty of Engineering, Oslo University College, Norway

This manual corresponds to CFENGINE Edition 2.2.1 for version 2.2.1 as last updated 4 May 2007.

1 Introduction to reference manual

The purpose of the cfengine reference manual is to collect together and document the raw facts about the different components of cfengine. Once you have become proficient in the use of cfengine, you will no longer have need of the tutorial. The reference manual, on the other hand, changes with each version of cfengine. You will be able to use it online, or in printed form to find out the details you require to implement configurations in practice.

1.1 Installation

In order to install cfengine, you should first ensure that the following packages are installed.

OpenSSL Open source Secure Sockets Layer for encryption.
URL: `http://www.openssl.org`

BerkeleyDB (version 3.2 or later)
Light-weight flat-file database system.
URL: `http://www.sleepycat.com`

The preferred method of installation is then

```
tar zxf cfengine-x.x.x.tar.gz
cd cfengine-x.x.x
./configure
make
make install
```

This results in binaries being installed in '`/usr/local/sbin`'. Since this is not necessarily a local file system on all hosts, users are encouraged to keep local copies of the binaries on each host, inside the cfengine trusted work directory.

1.2 Work directory

In order to achieve the desired simplifications, it was decided to reserve a private work area for the cfengine tool-set. In cfengine 1.x, the administrator could choose the locations of configuration files, locks, and logging data independently. In cfengine 2.x, this diversity has been simplified to a single directory which defaults to '`/var/cfengine`' (similar to '`/var/cron`'):

```
/var/cfengine
/var/cfengine/bin
/var/cfengine/inputs
/var/cfengine/outputs
```

The installation location '`/usr/local/sbin`' is not necessarily a local file system, and cannot therefore be trusted to a) be present, and b) be authentic on an arbitrary system.

Similarly, a trusted cache of the input files must now be maintained in the '`inputs`' subdirectory. When cfengine is invoked by the scheduler, it reads only from this directory. It is up to the user to keep this cache updated, on each host. This simplifies and consolidates the cfengine resources in a single place. The environment variable `CFINPUTS` still overrides this default location, as before, but in its absence or when called from the scheduler, this becomes the location of trusted files. A special configuration file '`update.conf`' is parsed and run before the main configuration is parsed, which is used to ensure that the currently caches policy is up-to-date. This has private classes and variables. If no value is set for `CFINPUTS`, then the default location is the trusted cfengine directory '`/var/cfengine/inputs`'.

The '`outputs`' directory is now a record of spooled run-reports. These are mailed to the administrator, as previously, or can be copied to another central location and viewed in an alternative browser..

1.3 Cfengine hard classes

A single class can be one of several things:

- The name of an operating system architecture e.g. `ultrix`, `sun4`, etc. This is referred to as a *hard class*.
- The unqualified name of a particular host. If your system returns a fully qualified domain name for your host, cfagent truncates it at the first dot.
- The name of a user-defined group of hosts.
- A day of the week (in the form `Monday, Tuesday, Wednesday, ..`).
- An hour of the day (in the form `Hr00, Hr01 ... Hr23`).
- Minutes in the hour (in the form `Min00, Min17 ... Min45`).
- A five minute interval in the hour (in the form `Min00_05, Min05_10 ... Min55_00`)
- A day of the month (in the form `Day1 ... Day31`).
- A month (in the form `January, February, ... December`).
- A year (in the form `Yr1997, Yr2004`).
- An arbitrary user-defined string.
- The IP address octets of any active interface (in the form `ipv4_192_0_0_1`, `ipv4_192_0_0`, `ipv4_192_0`, `ipv4_192`).

To see all of the classes define on a particular host, run

```
host# cfagent -p -v
```

as a privileged user. Note that some of the classes are set only if a trusted link can be established with cfenvd, i.e. if both are running with privilege, and the '`/var/cfengine/env_data`' file is secure.

1.4 Filenames and paths

Filenames in Unix-like operating systems

The directory separator is the forward slash '/' character. All references to file locations must be absolute pathnames in cfengine, i.e. they must begin with a complete specification of which directory they are in. For example:

```
/etc/passwd
/usr/local/masterfiles/distfile
```

The only place where it makes sense to refer to a file without a complete directory specification is when searching through directories for different kinds of file, e.g.

```
tidy:

  /home/user pattern=core age=0 recurse=inf
```

Here, one can write '`core`' without a path, because one is looking for any file of that name in a number of directories.

Cfengine was implemented primarily on Unix-like operating systems, but has since been ported to Windows operating systems and MacOS X. The Windows operating systems traditionally use a different filename convention. The following are all valid absolute file names under Windows:

```
c:\winnt
c:/winnt
/var/cfengine/inputs
//fileserver/share2/dir
```

The 'drive' name "C:" in Windows refers to a partition or device. Unlike Unix, Windows does not integrate these seamlessly into a single file-tree. This is not a valid absolute filename:

```
\var\cfengine\inputs
```

Paths beginning with a backslash are assumed to be win32 paths. They must begin with a drive letter or double-slash server name.

1.5 Debugging with signals

It is possible to turn debugging output on or off on a running cfagent. This is useful for troubleshooting the cause of hangups, or for getting debugging output from a cfagent launched from cfexecd.

A running cfagent process that receives a SIGUSR1 will immediately begin to behave as if it had been invoked with the '-d2' option. A SIGUSR2 will cause a running cfagent to run as if the '-d2' option had not been invoked.

Note that this output is often quite verbose.

2 Cfkey reference

The very first thing you should do on every host is to establish a public-private key pair. To do this, you need to run the program

```
everyhost# cfkey
```

on every host. This program needs to produce random numbers, and needs a source of randomness. A good strategy is to install and run the 'cfenvd' program for a week or two in advance of deploying cfengine 2, since 'cfenvd' collects random events, which are an excellent source of entropy for random number generation.

If you get the error message "PRNG not seeded", it means that insufficient data were found in order to make a random key. In that case, run 'cfenvd' for a few days more and try again.

3 Cfshow reference

The `cfshow` command was introduced in cfengine 2.1.11 in order to provide a simple way to show some of the data stored by cfagent for operational purposes.

```
everyhost# cfshow -a
everyhost# cfshow -l
everyhost# cfshow -c
everyhost# cfshow -s
everyhost# cfshow -p
```

The command line options are

'`-a --active`'
: This prints a list of any currently active locks, i.e. tasks that cfengine believes it is currently enagaged in.

'`-c --checksum`'
: This lists all of the files and their current checksum values in the current checksum database.

'`-l --locks`'
: This prints a list of the locks and the last times an active lock was secured for each cfengine acivity. This list is potentially very long.

'`-s --last-seen`'
: This lists the IP addresses of all known peers and the times they were last engaged in communication with the current host. The expected interval between communications is also printed. See FriendStatus. The output format is in a form that can easily be parsed by user scripts. e.g.

```
IP + 192.168.1.101 192.168.1.101   [Tue Jan 23 16:13] not seen for (6.42) hrs, Av 0.02 +/-
0.01 hrs
IP - 192.168.1.101 192.168.1.101   [Tue Jan 23 16:13] not seen for (6.42) hrs, Av 0.02 +/-
0.01 hrs
```

Lines marked with a `+` are attempts to connect by cfagent, lines with a `-` are connections into cfservd.

'`-s --performance`'
: Shows the time in seconds required to complete copies and shell executions.

```
(0.00 mins  Tue Feb 13 19:05) Av 0.00 +/- 0.00 for Copy(localhost:/usr/local/sbin/cfagent > /var/cfengine/bin/cfagent)
(0.00 mins  Tue Feb 13 19:05) Av 0.00 +/- 0.00 for Copy(localhost:/usr/local/sbin/cfenvd > /var/cfengine/bin/cfenvd)
(0.02 mins  Tue Feb 13 19:05) Av 0.02 +/- 0.00 for Copy(localhost:/usr/local/sbin/cfexecd > /var/cfengine/bin/cfexecd)
(0.00 mins  Tue Feb 13 19:05) Av 0.00 +/- 0.00 for Copy(localhost:/usr/local/sbin/cfservd > /var/cfengine/bin/cfservd)
(6.41 mins  Tue Feb 13 18:50) Av 0.00 +/- 0.00 for Exec(/usr/bin/updatedb -
-prunepaths=/media)
(0.00 mins  Tue Feb 13 19:05) Av 0.00 +/- 0.00 for Exec(/usr/sbin/ntpdate 128.39.74.16 > /dev/null)
```

4 Cfagent reference

4.1 Cfagent intro

Cfagent is the workhorse of cfengine. It interprets and computes the necessary strategies for implementing convergent maintenance. In order to carry out work efficiently, the agent groups similar actions together. The order of these actions is goverened by a list called the actionsequence.

In many cases, cfagent will be able to complete all its work in a single pass of the actionsequence. However, in complex configurations, it is hard to resolve all of the ordering dependencies automatically in a single pass. Cfagent keeps track both of all actions that have been performed and of those that might still need to be performed (given that some actions depend on the later outcomes of others). If there is a possibility that an action ordering dilemma might occur, it runs a second pass of the actionsequence to more quickly resolve the dependency (avoiding the wait for next scheduled run). No actions are performed twice however, since the agent checks off actions that have already been performed to avoid unnecessary duplication.

4.1.1 The file cfagent.conf

```
control:

  classes::

      domain = ( DNS-domain-name )

classes:

  Class/Group definitions

import:

  Files to import

# other items
```

4.1.2 Cfagent runtime options

Note that GNU long options are available with the syntax `--longoption`. The long names are given in brackets.

‘-a’ (`--sysadm`) Print only the name of the system administrator then quit.

‘-A’ (`--auto`) Can be used to signify an automatic run of cfengine, as opposed to a manual run. The distinction is not predetermined. Use of this option currently causes cfengine to ignore locks. This option is reserved for future development.

‘-b’ (`--force-net-copy`) Normally cfengine detects attempts to copy from a server via the network that will loop back to the localhost. It then avoids using the network to make the copy. This option forces cfengine to copy using the network. *Yes, someone thinks this is useful!*

'-c' (`--no-check-files`) Do not check file systems for ownership / permissions etc.

'-C' (`--no-check-mounts`) Check mount points for consistency. If this option is specified then directories which lie in the "mount point" area are checked to see whether there is anything mounted on them. Normally this is *off* since not all machines use mounted file systems in the same way. e.g. HPUX does not generally operate with partitions, but nevertheless one might wish to mimick a partition-like environment there, but it would be irritating to be informed that nothing was mounted on the mount point.

'-d' (`--debug`) Enable debugging output. Normally you will want to send this to a file using the shell script command or a pipe. -d1 shows only parsing output. -d2 shows only runtime action output. -d0 shows both levels. Debugging ouput is intended mainly for the author's convenience and is not a supported feature. The details of this output may change at any time.

'-D' (`--define`) Define a compound class symbol of the form *alpha.beta.gamma*.

'-e' (`--no-edits`) Suppress file editing.

'-E' (`--enforce-links`) Globally force links to be created where plain files or links already exist. Since this option is a big hammer, you have to use it in interactive mode and answer a yes/no query before cfengine will run like this.

'-f' (`--file`) Parse filename after this switch. By default cfengine looks for a file called *cfengine.conf* in the current directory.

'-h' (`--help`) Help information. Display version banner and options summary.

'-H' (`--no-hard-classes`). Prevents cfengine from generating any built-in class name information. Can be used for emulation purposes.

'-i' (`--no-ifconfig`) Do not attempt to configure the local area network interface.

'-I' (`--inform`) Switches on the inform output level, whereby cfengine reports everything it changes..

'-k' (`--no-copy`) Do not copy/image any files.

'-K' (`--no-lock`) Ignore locks when running.

'-l' (`--traverse-links`) Normally cfengine does not follow symbolic links when recursively parsing directories. This option will force it to do so.

'-L' (`--delete-stale-links`) Delete links which do not point to existing files (except in user home directories, which are not touched).

'-m' (`--no-mount`) Do not attempt to mount file systems or edit the filesystem table.

'-M' (`--no-modules`) Ignore modules in actionsequence.

'-n' (`--recon`,`--dry-run`,`--just-print`) No action. Only print what has to be done without actually doing it.

'-N' (`--negate`,`--undefine`) Cancel a set of classes, or undefine (set value to *false*) a compound class of the form *alpha.beta.gamma*.

'-p' (`--parse-only`) Parse file and then stop. Used for checking the syntax of a program. You do not have to be superuser to use this option.

'-P' (`--no-processes`) Do not execute the processes action.

'-q' (`--no-splay`) Switch off host splaying (sleeping).

'-Q' (`--quert`) Query the values of the comma separated list of variable names.

'-s' (`--no-commands`) Do not execute scripts or shell commands.

'-S' (`--silent`) Silence run time warnings.

'-t' (`--no-tidy`) Do not tidy file systems.

'-u' (`--use-env`) Causes cfengine to generate an environment variable '`CFALLCLASSES`' which can be read by child processes (scripts). This variable contains a summary of all the currently defined classes at any given time. This option causes some System V systems to generate a Bus Error or segmentation fault. The same information is available from the cfengine built-in variable `$(allclasses)` and can be passed as a parameter to scripts.

'-U' (`--underscore-classes`). When this option is set, cfengine adds an underscore to the beginning of the hard system classes (like `_sun4`, `_linux` etc. The longer compound classes are not underscored, since these are already complex and would unlikely result in collisions.) This can be used to avoid naming conflicts if you are so unjudicious as to name a host by the name of a hard class. Other classes are not affected.

'-v' (`--verbose`) Verbose mode. Prints detailed information about actions and state.

'-V' (`--version`) Print only the version string and then quit.

'-x' (`--no-preconf`) Do not execute the '`cf.preconf`' net configuration file.

'-X' (`--no-links`) Do not execute the `links` section of a program.

'-w' (`--no-warn`,`--quiet`) Do not print warning messages.

'-z' (`--schedule`) Print the exec schedule for the LAN (used by cfexecd).

In version 2.0.4, an abbreviation for actionsequence exclusions was added:

```
$ cfagent --avoid resolve,copy
$ cfagent --just tidy --just shellcommands
```

4.2 Variable expansion and contexts

Variables in cfengine 2 are defined in contexts. Variables in a given context refer to the different phases of execution of cfengine: global, update and main. In the "current" context, variables have the form

```
$(variable) ${variable}
```

and are expanded either on parsing or at execution. Variables that cannot be expanded remain as dollar strings. Variables belonging to a context that is not the current one may be referred to as

```
$(context.variable)  or  ${context.variable}
```

There is no difference between these forms as far as cfengine is concered. Some authors like to use the () form for cfengine variables, to distinguish them with shell variables in command strings. When using the () form in function arguments, they should be quoted to avoid parsing errors.

Consider the example:

```
$(global.env_time)
```

Some variables in cfengine are associative arrays (as made famous by Perl). Such arrays are referred to by square brackets:

```
$(array[key])  $(array[$(key)])
```

and so on. Note carefully that cfengine requires parentheses or braces around variable names. Unlike in the shell, they cannot be omitted.

4.2.1 Setting variables with functions

A number of special functions can be used to set variables in cfengine. You can import values from the execution of a shell command by prefixing a command with the word `exec`. This method is deprecated as of cfengine version 2; use the `ExecResult` function instead.

```
control:

 # old method

 listing = ( "exec /bin/ls -l" )

 # new method

 listing = ( ExecResult(/bin/ls -l) )
```

This sets the variable 'listing' to the output of the command in the quotes.

Some other built-in functions are

`A(X,Y)` Makes an associative array entry, associating X and Y. For instance:

```
control:

 assoc_array = ( A(B,"is for bird") A(C,"is for cat") )
```

results in:

```
OBJECT: main
   4569 : assoc_array[B]=is for bird
   4630 : assoc_array[C]=is for cat
```

Another example:

```
control:

 binhost = ( A(linux,machine1) A(solaris,machine2) )

copy:

 # Contact machine 1 for linux
 # Contact machine 2 for solaris

 /etc/source dest=/etc/receve server=$(binhost[$(class)])
```

`ExecResult(`*`command`*`)`
: Executes the named command without a shell-wrapper and inserts the output into the variable. Note that, when this is used in cfengine built-in list variables, any spaces are interpreted as list separators. In other lists, normal rules for iteration apply.

`ExecShellResult(`*`command`*`)`
: Executes the named command with a shell-wrapper and inserts the output into the variable. Note that, when this is used in cfengine built-in list variables, any spaces are interpreted as list separators. In other lists, normal rules for iteration apply.

`RandomInt(`*`a`*`,`*`b`*`)`
: Generate a random integer between a and b.

`ReadArray(`*`filename`*`,`*`fileformat`*`,`*`separator`*`,`*`comment`*`,`*`Max number of bytes`*`)`
: Reads up to a maximum number of bytes from a properly formatted file into a one-dimensional associated array. File formats are:

  `autokey` If this format is specified, `ReadArray` tries to interpret the file as a table of items separated with the separator character. Blank lines and comments (to end of line) are ignored. Items are keyed numerically starting from 1 to the maximum number in the file. The newline `$(n)` is always considered to be a separator, no matter what the current separator is.

  `textkey` If this format is specified, `ReadArray` tries to interpret the file as a list of lines of the form:

  ```
  key,value
  ```

`ReadFile(`*`filename`*`,`*`Max number of bytes`*`)`
: Read a maximum number of bytes from a file.

`ReadTable(`*`filename`*`,`*`fileformat`*`,`*`separator`*`,`*`comment`*`,`*`Max number of bytes`*`)`
: Reads up to a maximum number of bytes from a properly formatted file into a two-dimensional associated array.

  `autokey` If this format is specified, `ReadArray` tries to interpret the file as a table of items separated with the separator character. Blank lines and comments (to end of line) are ignored. Items are keyed numerically starting from 1 to the maximum number in the file. Any lines that do not contain the correct number of separators cause the function to fail without making any assignment.

`textkey` If this format is specified, `ReadArray` tries to interpret the file as a list of lines of the form:

```
key1,key2,value1
key3,key4,value2
```

This variable would then be references as $(table[key1][key2]).

`ReadList(`*`filename`*`,`*`fileformat`*`,`*`comment`*`,`*`Max number of bytes`*`)`
Reads up to a maximum number of bytes from a properly formatted file into a listvariable. File formats are:

`lines` If this format is specified, `ReadList` tries to interpret the file as a list of items on separate lines. The value returned is a list formatted by the `Split` character.

```
hosts = ( ReadList(/var/cfengine/inputs/datafile,lines,#,1000) )
```

`ReadTCP(`*`host/IP`*`,`*`portnumber`*`,`*`send string`*`,`*`Max number of bytes`*`)`
Reads up to a maximum number of bytes from a TCP service. Can be used to test whether certain services are responding to queries. It is recommended that this be used primarily to check services running on localhost. Bear in mind that this clause, executed on 100 hosts will produce the effect of a distributed denial of service attack, so the probe should be restricted to a single representative tester-host. For example:

```
one_host_only::

 # USE WITH CAUTION !

 probewww = ( ReadTCP(localhost,80,'GET index.html',1000) )
```

Or testing a network service:

```
control:

 checkhost::

 probesmtp = ( ReadTCP(localhost,25,"",1024) )
 probewww  = ( ReadTCP(project.iu.hio.no,80,"GET /viewcvs HTTP/1.0 ${n}${n}",1024) )

classes:

  viewcvs_error = ( RegCmp(".*Python Traceback.*","${probewww}") )

alerts:

  viewcvs_error::

     "Received viewcvs error from web server"
```

`SelectPartitionNeighbours(`*`filename`*`,`*`comment`*`,`*`Policy`*`,`*`group size`*`)`
This function is for use in peer to peer monitoring applications. It allows individual hosts to identify themselves as part of a group and find their peers.

The function returns a list variable, delimited by the list separation character, for use with Split.

```
control:

  allpeers = ( SelectPartitionNeighbours(/var/cfengine/inputs/cfrun.hosts,#,random,4) )

copy:

  /data/file dest=/p2prepository/file server=$(allpeers)
```

SelectPartitionLeader(*filename*,*comment*,*Policy*,*group size*)

This function is for use in peer to peer monitoring applications. It allows individual hosts to identify themselves as part of a group and select a leader. This function reads a text file of hostnames or IP addresses, one host per line, with blank lines and comments and partitions it into groups of a fixed size. It then returns picks a leader for the the group and returns its name as the value of the function.

```
control:

  leader = ( SelectPartitionLeader(/var/cfengine/inputs/cfrun.hosts,#,random,4) )

copy:

  /data/file dest=/p2prepository/file server=$(leader)
```

Note that functions should have no spaces between the function name and the leading parenthesis, but should themselves be surrounded by white space. For example:

```
control:

 variable2 = ( RandomInt(0,23) )

 variable3 = ( ExecResult(/bin/ls -a /opt) )

 myexcerpt  = ( ReadFile("/etc/services",220) )

 listvar = ( ReadArray(/tmp/array,textkey,",","#",100) )
```

In the latter case, the file could look like this:

```
host$ more /tmp/array
one,String to tbe read
two,Nothing string
three,Everything comes in threes
```

and results in the definition of (verify with `cfagent -p -d3`):

```
OBJECT: main
   960 : listvar[one]=String to tbe read
   259 : listvar[two]=Nothing string
   224 : listvar[three]=Everything comes in threes
```

4.2.2 Special variables

Variables are referred to in either of two different ways, depending on your taste. You can use the forms `$(variable)` or `${variable}`. The variable in braces or parentheses can be the name of any user defined macro, environment variable or one of the following special built-in variables.

`AllClasses`
: A long string in the form '`CFALLCLASSES=class1:class2...`'. This variable is a summary of all the defined classes at any given time. It is always kept up to date so that scripts can make use of cfengine's class data.

`arch`
: The current detailed architecture string—an amalgamation of the information from *uname*. *A constant*.

`binserver`
: The default server for binary data. *A constant*.

`ChecksumDatabase`
: If set to the name of a file, cfagent will use this to store checksums of important files, and give 'tripwire functionality', See Section 4.9.10 [ChecksumDatabase], page 39.

`ChecksumUpdates`
: If set to 'on', security information is automatically updated, See Section 4.9.12 [ChecksumUpdates], page 40.

`class`
: The currently defined system hard-class (e.g. `sun4`, `hpux`). *A constant*.

`date`
: The current date string. Note that if you use this in a shell command it might be interpreted as a list variable, since it contains the default separator '`:`'.

`domain`
: The currently defined domain.

`EmailFrom`
: The email address from whom email from cfexecd should appear to originate.

`EmailMaxLines`
: Most lines of output to email from a single cfexecd-induced run of cfagent. If undefined, defaults to 100. If set to 0, no email is sent by cfexecd. If set to `inf`, no maximum is enforced.

`EmailTo`
: The E-mail address to whom mail should be sent (overrides `sysadm` variable).

`faculty`
: The faculty or site as defined in control (see `site`).

`fqhost`
: The fully qualified hostname of the system.

`host`
: The hostname of the machine running the program.

`ipaddress`
: The numerical form of the Internet address of the host currently running cfengine found by a reverse lookup in DNS.

`ipv4[`*`interface`*`]`
: The IPv4 address of the named interface as determined from a probe of the interfaces. This variable belongs in the global context and refers to as in the following examples:

```
${global.ipv4[hme0]}
${global.ipv4[eth0]}
```

`MaxCfengines`
: The maximum number of cfengines which should be allowed to run concurrently on the system. This can prevent excessive load due to unintentional spamming in situations where several cfengines are started independently. The default value is unlimited.

`ostype`
: A short for of `$(arch)`.

`OutputPrefix`
: This quoted string can be used to change the default 'cfengine:' prefix on output lines to something else. You might wish to shorten the string, or have a different prefix for different hosts. The value in this variable is appended with the name of the host. The default is equivalent to,

```
OutputPrefix = ( "cfengine:$(host):")
```

`RepChar`
: The character value of the string used by the file repository in constructing unique filenames from path names. This is the character which replaces '`/`'.

`site`
: This variable is identical to `$(faculty)` and may be used interchangeably.

`smtpserver`
: The name of the host to which mail output should be sent.

`split`
: The character on which list variables are split.

`sysadm`
: The name or mail address of the system administrator.

`timezone`
: The current timezone as defined in `control`.

`UnderscoreClasses`
: If this is set to 'on' cfengine uses hard classes which begin with an underscore to avoid name collisions. See also the section Runtime Options.

`version`
: The current version string as defined in the code.

`year`
: The current year.

These variables are kept special because they play a special role in setting up a system configuration. You are encouraged to use them to define fully generalized rules in your programs. Variables can be used to advantage in defining filenames, directory names and in passing arguments to shell commands. The judicious use of variables can reduce many definitions to a single one if you plan carefully.

> *NOTE: the above control variables are not case sensitive, unlike user macros, so you should not define your own macros with these names.*

The following variables are also reserved and may be used to produce troublesome special characters in strings.

`cr`
: Expands to the carriage return character.

`colon` Expands to the colon ':' character.

`dblquote` Expands to a double quote "

`dollar` Expands to '$'.

`lf` Expands to a line-feed character (Unix end of line).

`n` Expands to a newline character.

`quote` Expands to a single quote '.

`spc` Expands simply to a single space. This can be used to place spaces in filenames etc.

`tab` Expands to a single tab character.

4.2.3 Iteration over lists

Variables can be used as iterators in some situations. Iteration over lists is currently rather limited in cfengine and is something to be improved on in a future version. When a variable is used as an iterator, a character is chosen to represent a list separator, as in the shell 'IFS' variable. The default separator is the colon ':' character:

```
control:

 listvar = ( one:two:three:four )
```

The action that contains a variable to be interpreted as a list appears as separate actions, one for each case:

```
shellcommand:

  "/bin/echo $(listvar)"
```

is equivalent to

```
shellcommand:

  "/bin/echo one"
  "/bin/echo two"
  "/bin/echo three"
  "/bin/echo four"
```

If multiple iterators are used, these are handled as nested loops:

```
cfengine::/bin/echo one 1:     one 1
cfengine::/bin/echo one 2:     one 2
cfengine::/bin/echo one 3:     one 3
cfengine::/bin/echo one 4:     one 4
cfengine::/bin/echo two 1:     two 1
cfengine::/bin/echo two 2:     two 2
cfengine::/bin/echo two 3:     two 3
cfengine::/bin/echo two 4:     two 4
cfengine::/bin/echo three:     three 1
cfengine::/bin/echo three:     three 2
cfengine::/bin/echo three:     three 3
cfengine::/bin/echo three:     three 4
cfengine::/bin/echo four :     four 1
cfengine::/bin/echo four :     four 2
cfengine::/bin/echo four :     four 3
cfengine::/bin/echo four :     four 4
```

Where iterators are not allowed, the implied lists are treated as scalars:

```
alerts:

 amnexus::

  "do $(list1) $(list2)"
```

e.g.

```
cfengine:: do one:two:three:four 1:2:3:4
```

Iterative expansion is currently restricted to:

- In the directory field of the admit/deny server access rules,
- In the 'from' field of a copy action,
- In the server field of the copy action,
- In the directory field of the disable action,
- In the directory field of the files action,
- In the 'to' field of a multiple link action,
- In the directory field of the required/disk action,
- In a resolve item.
- In the directory field of a tidy action,
- Names in the ignore action.

4.3 Cfengine classes

A *cfengine action* looks like this:

```
action-type:

   compound-class::

       declaration
```

A single class is an identifier that may consist of any alphanumeric character or the underscore, just like identifiers in ordinary programming languages. Classes that are derived from data like IP addresses or host names convert any other characters (like '`.`' or '`-`') into underscores. A single class can be one of several things:

- The name of an operating system architecture e.g. `ultrix`, `sun4`, etc. This is referred to as a *hard class*.
- The unqualified name of a particular host. If your system returns a fully qualified domain name for your host, cfagent truncates it at the first dot.
- The name of a user-defined group of hosts.
- A day of the week (in the form `Monday, Tuesday, Wednesday, ..`).
- An hour of the day (in the form `Hr00, Hr01 ... Hr23`).
- Minutes in the hour (in the form `Min00, Min17 ... Min45`).
- A five minute interval in the hour (in the form `Min00_05, Min05_10 ... Min55_00`)
- A quart hour (in the form `Q1, Q2, Q3, Q4`)
- An abbreviated time with quarter hour specified (in the form `Hr00_Q1, Hr23_Q4` etc.)

- A day of the month (in the form `Day1 ... Day31`).
- A month (in the form `January, February, ... December`).
- A year (in the form `Yr1997, Yr2004`).
- An arbitrary user-defined string.
- The IP address octets of any active interface (in the form `ipv4_192_0_0_1`, `ipv4_192_0_0`, `ipv4_192_0`, `ipv4_192`).

A compound class is a sequence of simple classes connected by dots or 'pipe' symbols (vertical bars). For example:

```
myclass.sun4.Monday::

sun4|ultrix|osf::
```

A compound class evaluates to 'true' if all of the individual classes are separately true, thus in the above example the actions which follow `compound_class::` are only carried out if the host concerned is in `myclass`, is of type `sun4` and the day is Monday! In the second example, the host parsing the file must be either of type `sun4` *or* `ultrix` *or* `osf`. In other words, compound classes support two operators: AND and OR, written '`.`' and '`|`' respectively. From cfengine version 2.1.1, I bit the bullet and added '`&`' as a synonym for the AND operator. Cfengine doesn't care how many of these operators you use (since it skips over blank class names), so you could write either

```
solaris|irix::
```

or

```
solaris||irix::
```

depending on your taste. On the other hand, the order in which cfengine evaluates AND and OR operations *does* matter, and the rule is that AND takes priority over OR, so that '`.`' binds classes together tightly and all AND operations are evaluated before ORing the final results together. This is the usual behaviour in programming languages. You can use round parentheses in cfengine classes to override these preferences.

Cfengine allows you to define switch on and off dummy classes so that you can use them to select certain subsets of action. In particular, note that by defining your own classes, using them to make compound rules of this type, and then switching them on and off, you can also switch on and off the corresponding actions in a controlled way. The command line options `-D` and `-N` can be used for this purpose. See also `addclasses` in the Reference manual.

A logical NOT operator has been added to allow you to exclude certain specific hosts in a more flexible way. The logical NOT operator is (as in C and C++) '`!`'. For instance, the following example would allow all hosts except for `myhost`:

```
action:

 !myhost::
```

```
          command
```

and similarly, so allow all hosts in a user-defined group `mygroup`, *except* for `myhost`, you would write

```
     action:

      mygroup.!myhost::

          command
```

which reads 'mygroup AND NOT myhost'. The NOT operator can also be combined with OR. For instance

```
     class1|!class2
```

would select hosts which were either in class 1, or those which were not in class 2.

Finally, there is a number of reserved classes. The following are hard classes for various operating system architectures. They do not need to be defined because each host knows what operating system it is running. Thus the appropriate one of these will always be defined on each host. Similarly the day of the week is clearly not open to definition, unless you are running cfengine from outer space. The reserved classes are:

```
  ultrix, sun4, sun3, hpux, hpux10, aix, solaris, osf, irix4, irix, irix64
     sco, freebsd, netbsd, openbsd, bsd4_3, newsos, solarisx86, aos,
            nextstep, bsdos, linux, debian, cray, unix_sv, GnU, NT
```

If these classes are not sufficient to distinguish the hosts on your network, cfengine provides more specific classes which contain the name and release of the operating system. To find out what these look like for your systems you can run cfengine in 'parse-only-verbose' mode:

```
    cfagent -p -v
```

and these will be displayed. For example, Solaris 2.4 systems generate the additional classes `sunos_5_4` and `sunos_sun4m`, `sunos_sun4m_5_4`.

Cfengine uses both the unqualified and fully host names as classes. Some sites and operating systems use fully qualified names for their hosts. i.e. `uname -n` returns to full domain qualified hostname. This spoils the class matching algorithms for cfengine, so cfengine automatically truncates names which contain a dot '.' at the first '.' it encounters. If your hostnames contain dots (which do not refer to a domain name, then cfengine will be confused. The moral is: don't have dots in your host names! *NOTE: in order to ensure that the fully qualified name of the host becomes a class you must define the domain variable.* The dots in this string will be replaced by underscores.

In summary, the operator ordering in cfengine classes is as follows:

'`()`' Parentheses override everything.

'`!`' The NOT operator binds tightest.

'`. &`' The AND operator binds more tightly than OR.

'`|`' OR is the weakest operator.

4.3.1 Setting classes with special functions

Cfengine provides a number of built-in functions for evaluating classes, based on file tests. Using these built-in functions is quicker than calling the shell `test` function. The time functions place their arguments in chronological order.

`IsNewerThan(`*`f1,f2`*`)`
: True if file 2 was modified more recently than file 1 (UNIX mtime)

`AccessedBefore(`*`f1,f2`*`)`
: True if file 1 was accessed more recently than file 2 (UNIX atime)

`ChangedBefore(`*`f1,f2`*`)`
: True if file 1's attributes were changed in any way more recently than file 2's (UNIX ctime)

`FileExists(`*`file`*`)`
: True if the named file object exists.

`IPRange(`*`address-range`*`)`
: True if the current host lies within the specified IP range

`HostRange(`*`basename,start-stop`*`)`
: True if the current relative domain name begins with basename and ends with an integer between start and stop. Note well: matching is case insensitive (both hostname and basename are converted to all upper case for comparison.)

`IsDefined(`*`variable-id`*`)`
: True if the named variable is defined. Note well: use the variable name, not its contents (that is, `IsDefined(var)`, and not `IsDefined(${var})`)

`IsDir(`*`f`*`)`
: True if the file f is a directory

`IsLink(`*`f`*`)`
: True if the file f is a symbolic link

`IsPlain(`*`f`*`)`
: True if the file f is a plain file

`PrepModule(`*`module,arg1 arg2...`*`)`
: True if the named module exists and can be executed. The module is assumed to follow the standard programming interface for modules (see Writing plugin modules in tutorial). Unlike actionsequence modules, these modules are evaluated immediately on parsing. Note that the module should be specified relative to the authorized module directory.

`Regcmp(`*`regexp,string or list separated string`*`)`
: True if the string matched the regular expression regexp.

`ReturnsZero(`*`command`*`)`
: True if the named shell command returns with exit code zero (success). The command is executed without a shell wrapper.

`ReturnsZeroShell(`*`command`*`)`
: True if the named shell command returns with exit code zero (success) when executed in the environment of a shell wrapper.

`Strcmp(s1,s2)`
: True if the string s1 exactly matches s2

`IsGreaterThan(s1,s2)`
: Returns true if the value of s1 is greater than the value of s2. Note that, if the strings have numerical values, a numerical comparison is performed, otherwise a string comparison is used.

`IsLessThan(s1,s2)`
: Returns true if the value of s1 is less than the value of s2. Note that, if the strings have numerical values, a numerical comparison is performed, otherwise a string comparison is used.

```
control:

    actionsequence = ( files )

 a = ( 2.12 )
   b = ( 2.11 )

classes:

lt = ( IsLessThan(${a},${b}) )
    gt = ( IsGreaterThan(${a},${b}) )

alerts:

  lt:: "$(a) LESS THAN $(b)"
  gt:: "$(a) GREATER THAN $(b)"
```

For example:

```
classes:

 access_to_dir = ( ReturnsZero(/bin/cd /mydir) )
 compare       = ( ChangedBefore(/etc/passwd_master,/etc/passwd) )
 isplain       = ( IsPlain(/tmp/import) )
 inrange       = ( IPRange(128.39.89.10-15) )
 CIDR          = ( IPRange(128.39.89.10/24) )
 compute_nodes = ( HostRange(cpu-,01-32)
 gotinit       = ( PrepModule(startup2,"arg1 arg2") )
```

4.4 acl

```
acl:

   class::

      { acl-alias

      action
      }
```

Cfengine's `ACL` feature is a common interface for managing filesystem access control lists (ACLs). An access control list is an extended file permission. It allows you to open or close a file to a named list of users (without having to create a group for those users); similarly, it allows you to open or close a file for a list of groups. Several operating systems have access control lists, but each typically has a different syntax and different user interface to this facility, making it very awkward to use. This part of a cfengine configuration simplifies the management of ACLs by providing a more convenient user interface for controlling them and—as far as possible—a common syntax.

An ACL may, by its very nature, contain a lot of information. Normally you would set ACLs in a `files` command, See Section 4.17 [files], page 86, or a `copy` command, See Section 4.11 [copy], page 58. It would be too cumbersome to repeat all of the information in every command in your configuration, so cfengine simplifies this by first associating an alias together with a complex list of ACL information. This alias is then used to represent the whole bundle of ACL entries in a `files` or `copy` command. The form of an ACL is similar to the form of an `editfiles` command. It is a bundle of information concerning a file's permissions.

```
{ acl-alias

 method:overwrite/append
 fstype:posix/solaris/dfs/afs/hpux/nt

 acl_type:user/group:permissions
 acl_type:user/group:permissions
 ...
}
```

The name *acl-alias* can be any identifier containing alphanumeric characters and underscores. This is what you will use to refer to the ACL entries in practice. The method entry tells cfengine how to interpret the entries: should a file's ACLs be overwritten or only adjusted? Since the filesystems from different developers all use different models for ACLs, you must also tell cfengine what kind of filesystem the file resides on. Currently only Solaris and DCE/DFS ACLs are implemented.

NOTE: if you set both file permissions and ACLs the file permissions override the ACLs.

4.4.1 Access control entries

An access control list is build of any number of individual access control entries (ACEs). The ACEs has the following general syntax:

```
acl_type:user/group:permissions
```

The user or group is sometimes referred to as a *key*.

For an explanation of ACL types and their use, refer to your local manual page. However, note that for each type of filesystem, there are certain entries which must exist in an ACL. If you are creating a new ACL from scratch, you must specify these. For example, in Solaris ACLs you must have entries for `user`, `group` and `other`. Under DFS you need what DFS calls a `user_obj`, `group_obj` and an `other_obj`, and in some cases `mask_obj`. In cfengine syntax these are called `user:*:`, `other:*:` and `mask:*:`, as described below. If you are appending to an existing entry, you do not have to re-specify these unless you want to change them.

Cfengine can overwrite (replace) or append to one or more ACL entries.

`overwrite`
: `method:overwrite` is the default. This sets the ACL according to the specified entries which follow. The existing ACL will be overwritten completely.

`append`
: `method:append` adds or modifies one or more specified ACL entries. If an entry already exists for the specified type and user/group, the specified permission bits will be added to the old permissions. If there is no ACL entry for the given type and user/group, a new entry will be appended.

If the new ACL exactly matches the existing ACL, the ACL is not replaced.

The individual bits in an ACE may be either added subtracted or set equal to a specified mask. The '+' symbol means add, the '-' symbol subtract and '=' means set equal to. Here are some examples:

```
acltype:id/*:mask

user:mark:+rx,-w
user:ds:=r
user:jacobs:noaccess
user:forgiven:default

user:*:rw
group:*:r
other:*:r
```

The keyword `noaccess` means set all access bits to zero for that user, i.e. remove all permissions. The keyword `default` means remove the named user from the access crontrol list altogether, so that the default permissions apply. A star/asterisk in the centre field indicates that the user or group ID is implicitly specified as of the owner of the file, or that no ID is applicable at all (as is the case for 'other').

4.4.2 Solaris ACLs

Under Solaris, the ACL type can be one of the following:

```
user
group
mask
other
default_user
default_group
default_mask
default_other
```

A user or group can be specified to the user, group, default_user and default_group types. Solaris ACL permissions are the normal UNIX permissions bits '`rwx`', where:

```
r - Grants read privileges.
w - Grants write privileges.
x - Grants execute privileges.
```

4.4.3 DFS ACLs

In DCE, the ACL type can be one of the following:

```
other
mask
any
unauthenticated
user
group
foreign_other
foreign_user
foreign_group
```

The `user`, `group`, `foreign_user` and `foreign_group` types require that you specify a user or group. The DCE documentation refers to types `user_obj`, `group_obj` and so on. In the cfengine implementation, the ugly '`_obj`' suffix has been dropped to make these more in keeping with the POSIX names. `user_obj::`, is equivalent to `user:*:` is cfengine. The star/asterisk implies that the ACL applies to the owner of the file object.

DFS permissions are comprised of the bits '`crwxid`', where:

```
c - Grants control privileges, to modify an acl.
r - Grants read  privileges.
w - Grants write privileges.
x - Grants execute privileges.
i - Grants insert privileges.
d - Grants delete privileges.
```

See the DCE/DFS documentation for more information about this.

It is not possible to set ACLs in foreign cells currently using cfengine, but you can still have all of your ACL definitions in the same file. You must however arrange for the file to be executed on the server for the cell concerned. Note also that you must perform a DCE login (normally as user '`cell_admin`') in order to set ACLs on files which are not owned by the owner of the cfengine-process. This is because you must have a valid security ticket.

4.4.4 NT ACLs

NT ACEs are written as follows:

```
acl_type:user/group:permissions:accesstype
```

The actual change consists of the extra field containing the access type. A star/asterisk in the field for *user/group* would normally imply that the ACL applies to the owner of the file object. However this functionality is as of today not yet implemented.

In NT, the ACL type can be one of the following:

```
user
group
```

Both types require that you specify the name of a user or a group.

NT permissions are comprised of the bits '`rwxdpo`', where:

```
r - Read privileges
w - Write privileges
x - Execute privileges
d - Delete privileges
p - Privileges to change the permissions on the file
o - Privileges to take ownership of the file
```

In addition to any combination of these bits, the word `noaccess` or `default` can be used as explained in the previous section. NT comes with some standard, predefined permissions.

The standards are only a predefined combination of the different bits specified above and are provided with cfengine as well. You can use the standards by setting the permission to `read`, `change` or `all`. The bit implementation of each standard is as on NT:

```
read   - rx
change - rwxd
all    - rwxdpo
```

where the bits follow the earlier definition. The keywords mentioned above can only be used alone, and not in combination with '`+`', '`-`', '`=`' and/or other permission bits.

NT defines several different access types, of which only two are used in connection with the ACL type that is implemented in cfengine for NT. The access type can be one of the following:

```
allowed
denied
```

Intuitively, `allowed` access grants the specified permissions to the user, whilst `denied` denies the user the specified permissions. If no access type is specified, the default is `allowed`. This enables cfengine's behaviour as on UNIX systems without any changes to the configuration file. If the permissions `noaccess` or `default` is used, the access type will be irrelevant.

4.5 ACL Example

Here is an example of a configuration file for an NT ACL:

```
control:
        actionsequence = ( files )
        domain = ( iu.hioslo.no )

files:
        $(HOME)/tt    acl=acl_alias1    action=fixall

acl:
        { acl_alias1

        method:overwrite
        fstype:nt

        user:gustafb:rwx:allowed
        user:mark:all:allowed
        user:toreo:read:allowed
        user:torej:default:allowed
        user:ds2:+rwx:allowed

        group:dummy:all:denied
        group:iu:read:allowed
        group:root:all:allowed
        group:guest:dpo:denied
        }
```

4.5.1 ACL Example

Here is an example of a configuration file for one Solaris ACL and one DCE/DFS ACL:

```
control:
        actionsequence = ( files )
        domain = ( iu.hioslo.no )
```

```
files:
        $(HOME)/tt    acl=acl_alias1    action=fixall
        /:/bigfile    acl=acl_alias2    action=fixall

acl:
        { acl_alias1

        method:overwrite
        fstype:posix

        user:*:rwx
        user:mark:=rwx
        user:sowille:=rx
        user:toreo:=rx
        user:torej:default
        user:ds2:+rwx
        group:*:rx
        group:iu:r
        group:root:x
        mask:*:rx
        other:*:rx

        default_user:*:=rw
        default_user:mark:+rwx
        default_user:ds:=rwx
        default_group::=r
        default_group:iu:+r
        default_mask::w
        default_other::rwx
        }

        { acl_alias2

        method:overwrite
        fstype:dfs

        user:*:rwxcid
        group:*:rxd
        other:*:wxir
        mask:*:rxw
        user:/.../iu.hioslo.no/cell_admin:rc
        group:/.../iu.hioslo.no/acct-admin:rwxcid
        user:/.../iu.hioslo.no/root:rx
        }
```

4.6 alerts

Alerts are normally just messages that are printed when classes become activated in order to alert the system administrator to some condition that has arisen. Alerts can also be special functions, like `ShowState()` that generate system output.

Alerts cannot belong to the class `any`, that would generate a message from every host. In a huge network this could result in vast amounts of Email. This behaviour can be forced, however, by creating an alias for the class 'any' that is defined on the affected hosts.

```
alerts:
```

```
class::

   quoted message ifelapsed=time
   ShowState(parameter)
   SysLog(priority,message)
   SetState(name,ttl,policy)
   UnSetState(name)
   FriendStatus(hours)
   PrintFile('filename',lines)
```

For example:

```
alerts:

  myclass::

  "Reminder: say hello every hour" ifelapsed=60

  nfsd_in_high_dev2::

  "High NFS server access rate 2dev at $(host) value $(value_nfsd_in) av $(average_nfsd_in) pm $(stddev_nfsd_in)"
  ShowState(incoming.nfs)

 # ROOT PROCS

  anomaly_hosts.RootProcs_high_dev2::

   "RootProc anomaly high 2 dev on $(host) value $(value_rootprocs) av $(average_rootprocs) pm $(stddev_rootprocs)"
  ShowState(rootprocs)
```

The `ShowState()` function reports on state gathered by the cfenvd daemon.

```
ShowState(incoming.tcpsyn)
ShowState(outgoing.smtp)
ShowState(incoming.www)
ShowState(outgoing.www)
ShowState(procs)
ShowState(rootprocs)
ShowState(otherprocs)
ShowState(users)
```

To limit the frequency of alerts, you can set locking times:

```
 # ROOT PROCS

  anomaly_hosts.RootProcs_high_dev2::

   "RootProc anomaly high 2 dev on $(host) value $(value_rootprocs) av $(average_rootprocs) pm $(stddev_rootprocs)"

   ShowState(rootprocs)  ifelapsed=10 expireafter=20
```

Alerts can also be channeled directly to syslog, to avoid extraneous console messages or email.

```
  SysLog(LOG_ERR,"Test syslog message")
```

One application for alerts is to pass signals from one cfengine to another by persistent, shared memory. For example, suppose a short-lived anomaly event triggers a class that relates to a security alert. The event class might be too short-lived to be followed up by cfagent in full. One could thus set a long term class that would trigger up several follow-up checks. A persistent class could also be used to exclude an operation for an interval of time.

Persistent class memory can be added through a system alert functions to give timer behaviour. For example, consider setting a class that acts like a non-resettable timer. It is defined for exactly 10 minutes before expiring.

```
SetState("preserved_class",10,Preserve)
```

Or to set a class that acts as a resettable timer. It is defined for 60 minutes unless the SetState call is called again to extend its lifetime.

```
SetState(non_preserved_class,60,Reset)
```

Existing persistent classes can be deleted with:

```
UnsetState(myclass)
```

The `FriendStatus` function is available from version 2.1.4 and displays a message if hosts that normally have a cfengine protocol connection with the current host have not connected for more than than specified number of hours. If the number of hours is set to zero, cfengine uses a machine-learned expectation value for the time and uses this to report. The friend status of a host is thus the expectation that there is a problem with a remote peer. Expected contact rates of more than the variable `LastSeenExpireAfter` are ignored as spurious, See Section 4.9.40 [lastseenexpireafter], page 47.

The `PrintFile` function can be used to display short excerpts from text files. The arguments are the filename and a maximum number of lines to be printed.

4.7 binservers

The `binservers` declaration need only be used if you are using cfengine's model for mounting NFS filesystems. This declaration informs hosts of which other hosts on the network possess filesystems containing software (binary files) which client hosts should mount. This includes resources like programs in `/usr/local` and so on. A host may have several binary servers, since there may be several machines to which disks are physically attached. In most cases, on a well organized network, there will be only one *architecture server* per UNIX platform type, for instance a SunOS server, an ULTRIX server and so on.

Binary servers are defined as follows:

```
binservers:

   physics.sun4::   sunserver sunserver2
   physics.linux::  linuxserver
```

The meaning of this declaration is the following. All hosts of type `sun4` which are members of the group `physics` should mount any binaries declared in the `mountables` resource list which belong to hosts `sunserver` or `sunserver2`. Similarly all `linux` machines should mount binary filesystems in the mountables list from `linuxserver`.

Cfengine knows the difference between binaries and home directories in the `mountables` list, because home directories match the pattern given by `homepattern`. See Section 4.9.33 [homepattern], page 44. See Section 4.20 [homeservers], page 99.

Note that every host is a binary server for itself, so that the first binary server (and that with highest priority) is always the current host. This ensures that local filesystems are always used in preference to NFS mounted filesystems. This is only relevant in connection with the variable `$(binserver)`.

4.8 broadcast

This information is used to configure the network interface for each host.

Every local area network has a convention for determining which internet address is used for broadcast requests. Normally this is an address of the form `aaa.bbb.ccc.255` or `aaa.bbb.ccc.0`. The difference between these two forms is whether all of the bits in the last number are ones or zeroes respectively. You must find out which convention is used at your establishment and tell cfengine using a declaration of the form:

```
broadcast:

  any::

     ones     # or zeros, or zeroes
```

In most cases you can use the generic class `any`, since all of the hosts on the same subnet have to use the same convention. If your configuration file encompasses several different subnets with different conventions then you will need to use a more specific.

Cfengine computes the actual value of the broadcast address using the value specified above and the netmask See Section 4.9.46 [netmask], page 48.

4.9 control

The fundamental piece of any cfengine script or configuration file is the control section. If you omit this part of a cfengine script, it will not do anything! The control section is used to define certain variables, set default values and define the order in which the various actions you have defined will be carried out. Because cfengine is a declarative or descriptive language, the order in which actions appear in the file does not necessarily reflect the order in which they are executed. The syntax of declarations here is:

```
control:

   classes::

      variable = ( list or value function(args) )
```

The control section is a sequence of declarations which looks something like the following example:

```
control:

  site     = ( univ )
  domain   = ( univ.edu )
  sysadm   = ( admin@computing.univ.edu )
  netmask  = ( 255.255.252.0 )
  timezone = ( EDT )
  nfstype  = ( nfs )

  childlibpath = ( /usr/local:/mylibs )

  sensiblesize  = ( 1000 )
  sensiblecount = ( 2 )
  editfilesize  = ( 4000 )

  actionsequence =
     (
     links.some
     mountall
     links.others
     files
     )

  myvariable = ( something )
  mymacro    = ( somethingelse )
  myrandom   = ( RandomInt(3,6) )
  myexcerpt  = ( ReadFile("/etc/services",220))
```

Parentheses are required when making a declaring information in cfengine. Note that a limited number of built-in functions exists:

- `ExecResult`(*command*) Executes the named shell command and inserts the output into the variable. Note that, when this is used in cfengine built-in list variables, any spaces are interpreted as list separators. In other lists, normal rules for iteration apply.
- `RandomInt(a,b)` Is substituted for a random number between (a,b).

- `ReadFile`(*filename*,*Max number of bytes*) A maximum number of bytes is read from the named file and placed in a variable.

For more functions, See Section 4.2.1 [Setting variables with functions], page 12.

The meaning of each of these lines is described below.

4.9.1 AbortClasses

The `AbortClasses` list is a list of class identifiers that will result in the abortion of the current cfagent instanitation with an error message containing the name of the offending class.

```
AbortClasses = ( emergency nologin_exists )
```

This mechanism allows one to make controlled exceptions at the agent level. For example

```
control:

 actionsequence = ( shellcommands )

 AbortClasses = ( danger_will_robinson )

shellcommands:

  "shellcom 1"
  "shellcom 2" define=ok elsedefine=danger_will_robinson
```

4.9.2 access

The `access` list is a list of users who are to be allowed to execute a cfengine program. If the list does not exist then all users are allowed to run a program.

```
access = ( user1 user2 ... )
```

The list may consist of either numerical user identifiers or valid usernames from the password database. For example:

```
access = ( mark aurora 22 456 )
```

would restrict a script to users mark, aurora and user id 22 and 456.

4.9.3 actionsequence

The action sequence determines the order in which collective actions are carried out. Here is an example containing the full list of possibilities:

```
actionsequence =
   (
   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
   packages               # check for required packages
   shellcommands          # execute shell commands
   editfiles              # edit files
   addmounts              # add new filesystems to system
   directories            # make any directories
```

```
links                  # check and maintain links (single and child)
mailcheck              # check mailserver
mountall               # (again)
required               # check required filesystems
tidy                   # tidy files
disable                # disable files
files                  # check file permissions
copy                   # make a copy/image of a master file
processes              # signal / check processes
module:name            # execute a user-defined module
)
```

Here is a more complete description of the meaning of these keywords.

`addmounts`
: causes cfengine to compute which NFS filesystems are missing from the current host and add them. This includes editing the filesystem table, creating the mount-directory, if required. This command relies on information provided by `mountinfo`, so it should normally only be called after `mountinfo`. If the filesystem already appears to be in the filesystem table, a warning is issued.

`checktimezone`
: runs a check on the timezone defined for the shell running cfengine.

`directories`
: executes all the commands defined under the `directories` section of the program. It builds new directories.

`disable`
: executes all the commands defined under the `disable` section of the program.

`editfiles`
: executes all the commands defined under the `editfiles` section of the program.

`files`
: executes all the commands defined under the `files` section of the program.

`links`
: executes all the commands defined under the `links` section of the program.

`mailcheck`
: tests for the presence of the NFS-mounted mail spooling directory on the current host. The name of the mail spool directory is defined in the `mailserver` section of the cfengine program. If the current host is the same as the mailserver (the host which has the physical spool directory disk) nothing is done. Otherwise the filesystem table is edited so as to include the mail directory.

`module`
: Normally cfengine's ability to detect the systems condition is limited to what it is able to determine while excuting predefined actions. Classes may be switched on as a result of actions cfengine takes to correct a problem. To increase the flexibility of cfengine, a mechanism has been introduced in version 1.5 which allows you to include a module of your own making in order to define or undefine a number of classes. The syntax

```
module:mytests

"module:mytests arg1 arg2 .."
```

declares a user defined module which can potentially set the classes *class1* etc. Classes returned by the module must be declared so that cfengine knows to pay attention to rules which use these classes when parsing; this is done using `AddInstallable`. If arguments are passed to the module, the whole string must be quoted like a shellcommand. See ⟨undefined⟩ [Writing plugin modules], page ⟨undefined⟩. Whether or not these classes become set or not depends on the behaviour of your module. The classes continue to apply for all actions which occur after the module's execution. The module must be owned by the user executing cfengine or root (for security reasons), it must be named '`module:`*`module-name`*' and must lie in a special directory, See Section 4.9.44 [moduledirectory], page 47.

`mountall` mounts all filesystems defined in the hosts filesystem table. This causes new NFS filesystems added by `addmounts` and `mailcheck` to be actually mounted. This should probably be called both before `mountinfo` and after `addmounts` etc. A short timeout is placed on this operation to avoid hanging RPC connections when parsing NFS mounted file systems.

`mountinfo`
builds internal information about which filesystems are presently mounted on the current host. Cfengine assumes that required-filesystems which are not found need to be mounted. A short timeout is placed on this operation to avoid hanging RPC connections when parsing NFS mounted file systems. If this times out, no further mount operations are considered reliable and are summarily cancelled.

`netconfig`
checks the netmask, hostname, IP address and broadcast address for the current host. The correct values for the netmask and broadcast address are set if there is an error. The defaultroute is matched against the static routing table and added if no default route exists. This does not apply to DHCP clients, which set a default route automatically.

`required` executes all the commands defined under the `required` section of the program. It checks for the absence of important NFS resources.

`resolve` checks and corrects the DNS domain name and the order of nameservers in the file '`/etc/resolv.conf`'.

`packages` executes commands defined under the `packages` section of the program. This will query the system's package database for the specified packages, at the specified versions, set classes based on whether or not those packages exist, and optionally install those packages using a pre-defined package manager command.

`shellcommands`
executes all the commands defined under the `shellcommands` section of the program.

`tidy` executes all the commands defined under the `tidy` section of the program.

`unmount` executes all the commands defined under the `unmount` section of the program. The filesystem table is edited so as to remove the unwanted filesystems and the unmount operation is executed.

`processes`
: executes commands defined under the `processes` section of the program.

Under normal circumstances this coarse ordering is enough to suit most purposes. In some cases you might want to, say, only perform half the link operations before mounting filesystems and then, say, perform the remainder. You can do this (and similar things) by using the idea of defining and undefining classes. See ⟨undefined⟩ [Defining classes], page ⟨undefined⟩.

The syntax

```
actionsequence =
   (
   links.firstpass.include
   ...
   links.secondpass
   )
```

means that cfengine first executes `links` with the classes `firstpass` and `include` *defined*. Later it executes `links` with `secondpass` defined. You can use this method of adding classes to distinguish more finely the flow of control in programs.

A note about style: if you define and undefine lots of classes to do what you want to do, you might stop and ask yourself if your `groups` are defined as well as they should be. See Section 4.19 [groups], page 97. Programming in cfengine is about doing a lot for only a little writing. If you find yourself writing a lot, you are probably not going about things in the right way.

4.9.4 AddClasses

`AddClasses  = ( `*`list of identifiers`*` )`

The `AddClasses` directive is used to define a list of class attributes for the current host. Normally only the hard classes defined by the system are 'true' for a given host. It is convenient though to be able to define classes of your own to label certain actions, mainly so that they can later be excluded so as to cut short or filter out certain actions. This can be done in two ways. See Section 4.9.3 [actionsequence], page 34.

To define a list of classes for the current session, you write:

```
AddClasses = ( exclude shortversion )
```

This is equivalent to (though more permanent than) defining classes on the command line with the `-D` option. You can now use these to qualify actions. For example

```
any.exclude::
    ...
```

Under normal circumstances `exclude` is always true — because you have defined it to be so, but you can *undefine* it in two ways so as to prevent the action from being carried out. One way is to undefine a class on the command line when you invoke cfengine:

```
host#  cfengine -N exclude
```

or

```
host#  cfengine -N exclude.shortversion

host#  cfengine -N a.b.c.d
```

These commands run cfengine with the named classes *undefined*. That means that actions labelled with these classes are excluded during that run.

Another way to restrict classes is to add a list of classes to be undefined in the action-sequence. See next section.

4.9.5 AddInstallable

`AddInstallable  = ( list of identifiers )`

Some actions in your cfengine program will be labelled by classes which only become defined at run time using a `define=` option. Cfengine is not always able to see these classes until it meets them and tries to save space by only loading actions for classes which is believes will become defined at some point in the program. This can lead to some actions being missed if the action is parsed before the place where the class gets switched on, since cfengine is a one-pass interpreter,. To help cfengine determine classes which *might become defined* during a run, you can declare them in this list. It does no harm to declare classes here anyway. Here is an example where you need to declare a class because of the ordering of the actions.

```
control:

    AddInstallable  = ( myclass )

files:

   myclass::

     /tmp/test mode=644 action=fixall

copy:

     /tmp/foo dest=/tmp/test define=myclass
```

If we remove the declaration, then when cfengine meets the files command, it skips it because it knows nothing about the class '`myclass`'—when the copy command follows, it is too late. Remember that imported files are always parsed after the main program so definitions made in imported files always come later than things in the main program.

4.9.6 AllowRedefinitionOf

Normally cfagent warns about redefinitions of variables during parsing. This is presumed to be a mistake. To avoid this behaviour, add the name of the variable to this list, and the warning disappears.

```
control:

actionsequence = ( copy )

AllowRedefinitionOf = ( cfrep )

cfrep = ( bla )
```

```
cfrep = ( blo )
```

4.9.7 AutoDefine

```
control:

   hup_syslogd::

     autodefine = ( /etc/syslog.c* )
```

Referring to the class that prefixes the command, `autodefine` is a list of file patterns that will define the said class, if a named file is copied in any statement. This helps to avoid having to write a large number of file-specific copy: lines with define=class configured. In the example above, the class `hup_syslogd` would be defined if '`/etc/syslog.conf`' is copied at any time.

4.9.8 BinaryPaddingChar

```
BinaryPaddingChar = ( \0 )
```

This specifies the type of character used to pad strings of unequal length in `editfiles` during binary editing. The default value is the space character, since this is normally used to edit filenames or text messages within program code.

4.9.9 BindToInterface

If this is set to a specific IP address of an IP configured interface, cfagent will use that address for outgoing connections. On Multi-homed hosts this allows one to restrict the traffic to a known interface. An interface must be configured with an IP address in order to be bound.

This feature is not available for old operating systems.

4.9.10 ChecksumDatabase

```
ChecksumDatabase = ( /var/cfengine/cfdb )
```

If this filename is defined, cfengine will use it to store message digests (i.e. cryptographic checksums) of files for security purposes, See Section 4.17 [files], page 86, `checksum=`.

4.9.11 ChecksumPurge

```
ChecksumPurge = ( on )
```

This variable defaults to 'off'. If set to true, cfagent will look at all of the registered files in the database and check whether thet still exist. If the file no longer exists, it is removed from the database and a warning is issued.

To purge files now and then, but at no particular time, one could do something like this:

```
strategies:

 { purging

 NowAndThen: 1
 ElseWhen:   49
 }

control:

 NowAndThen::

   ChecksumPurge = ( on )
```

4.9.12 ChecksumUpdates

```
ChecksumUpdates = ( on )
```

This variable defaults to 'off'. If set to true, cfagent will automatically update the checksum of a file, if it changes on the disk. This means that a security warning will be issued only once about files which have changed, and the changed version will be re-registered as the correct version. This option could be switched on after a system upgrade, for instance, in order to update the database, and then switched to 'off' again to reduce the risk of missing a security alert. Alternatively, if you are confident that the first message is sufficient, it can be left as 'on' so that only one message is given.

4.9.13 CompressCommand

Specified the name of a program that is used in the files `action=compress` attribute.

```
CompressCommand = ( /usr/bin/gzip )
```

4.9.14 ChildLibPath

Sets a value for LD_LIBRARY_PATH in child processes:

```
 childlibpath = ( /usr/local/lib:/local/mysql/lib )
```

Note that the variables LD_LIBRARY_PATH is special. This library path is needed to run processes as children of cfengine. Often, if the agent is started from cron (which is started by init), there is no suitable library path set, and shellcommands will fail with strange errors about not being able to load shared objects. Setting a library path here is a useful way of correcting this problem.

4.9.15 CopyLinks

This list is used to define a global list of names or patterns which are to be copied rather than linked symbolically. For example

```
CopyLinks = ( *.config )
```

The same facility can be specified for each individual link operation using the `copy` option See Section 4.24 [links], page 103. Copying is performed using a file age comparison.

Note that all entries defined under a specified class are valid only as long as that class is defined. For instance

```
 class::

     CopyLinks = ( pattern )
```

would define a pattern which was only valid when *class* is defined.

4.9.16 DefaultCopyType

This parameter determines the default form of copying for all copy operations parsed after this variable. The legal values are `ctime` (intial default), `mtime`, `checksum` and `binary`. e.g.

```
DefaultCopyType = ( mtime )
```

4.9.17 DefaultPkgMgr

Sets the default value of the `pkgmgr` attribute for `packages` items.

```
DefaultPkgMgr = ( rpm )
```

By default, this variable is not set, meaning there will be no package manager selected, and each item in the `packages` section must specify its own package manager, or it will not be checked. For information on the values of this variable, See Section 4.30 [packages], page 123.

4.9.18 DeleteNonUserFiles

If this parameter is set to true, cfengine will delete files which do not have a name belonging to a known user id.

```
DeleteNonUserFiles = ( true )

SpoolDirectories = ( /var/spool/cron/crontabs )
```

This is an generalization of DeleteNonUserMail and makes it redundant. it is formally executed as a part of the “tidy” action.

4.9.19 DeleteNonOwnerFiles

If this parameter is set to true, cfengine will delete files on mailservers whose names do not correspond to a known user name, but might be owned by a known user.

```
DeleteNonOwnerFiles = ( true )

SpoolDirectories = ( /var/spool/cron/crontabs )
```

This is an generalization of DeleteNonOwnerMail and makes it redundant.

4.9.20 DeleteNonUserMail

If this parameter is set to true, cfengine will delete mail files on mailservers which do not have a name belonging to a known user id. This does not include lock files.

4.9.21 DeleteNonOwnerMail

If this parameter is set to true, cfengine will delete files on mailservers whose names do not correspond to a known user name, but might be owned by a known user.

4.9.22 domain

```
domain = ( domain name )
```

This variable defines the domainname for your site. You must define it here, because your system might not know its domainname when you run cfengine for the first time. The domainname can be used as a cfengine variable subsequently by referring to $(domain). The domainname variable is used by the action `resolve`. The domain is also used implicitly by other matching routines. You should define the domain as early as possible in your configuration file so as to avoid problems, especially if you have the strange practice of naming hosts with their fully qualified host names since groups which use fully qualified names can fail to be defined if cfengine is not able to figure out the domain name.

4.9.23 DPKGInstallCommand

Sets the command used to install packages that need to be installed under the DPKG package manager.

```
DPKGInstallCommand = ( "/usr/bin/pkgmgr %s" )
```

By default, this variable is not set, meaning that any packages with action=install will NOT be installed if installation is required. Note the `"`'s around the string, and the %s is replaced with the list of packages to be installed, each separated by a ' ' (space).

4.9.24 DryRun

```
DryRun = ( on/off )
```

This variable has the same effect as the command line options *--dry-run* or *-n*. It tells cfengine to only report what it should do without actually doing it.

```
classes::

  DryRun = ( on )
```

4.9.25 editbinaryfilesize

```
EditBinaryFileSize  = ( size )
```

Cfengine will refuse to edit a file which is larger than the value of `editbinaryfilesize` in bytes. This is to prevent possible accidents from occurring. The default value for this variable is 10000000 bytes. If you don't like this feature, simply set the value to be a very large number or to zero. If the value is zero, cfengine will ignore it.

4.9.26 editfilesize

```
EditfileSize  = ( size )
```

This variable is used by cfengine every time it becomes necessary to edit a file. Since file editing applies only to text files, the files are probably going to be relatively small in most cases. Asking to edit a very large (perhaps binary) file could therefore be the result of an error.

A check is therefore made as a security feature. Cfengine will refuse to edit a file which is larger than the value of `editfilesize` in bytes. This is to prevent possible accidents from occurring. The default value for this variable is 10000 bytes. If you don't like this feature, simply set the value to be a very large number or to zero. If the value is zero, cfengine will ignore it.

4.9.27 EmptyResolvConf

```
EmptyResolvConf  = ( true )
```

Normally cfengine does not tidy up old entries in the '`/etc/resolv.conf`' file. This option causes cfengine to remove all existing content from the file.

4.9.28 Exclamation

This variable defaults to "on". If set to "off", no exclamation marks (Br. pling, Am: shriek) are printed during security alerts, e.g. for checksum violations.

```
Exclamation  = ( off )
```

4.9.29 ExcludeCopy

This list is used to define a global list of names or patterns which are to be excluded from copy operations. For example

```
ExcludeCopy = ( *~ *% core )
```

The same facility can be specified for each individual link operation using the `exclude` option See Section 4.11 [copy], page 58.

Note that all entries defined under a specified class are valid only as long as that class is defined. For instance

```
class::

    ExcludeCopy = ( pattern )
```

would define a pattern which was only valid when *class* is defined.

4.9.30 ExcludeLink

This list is used to define a global list of names or patterns which are to be excluded from linking operations. For example

```
ExcludeLink = ( *~ *% core )
```

The same facility can be specified for each individual link operation using the `exclude` option See Section 4.24 [links], page 103.

Note that all entries defined under a specified class are valid only as long as that class is defined. For instance

```
class::

    ExcludeLink = ( pattern )
```

would define a pattern which was only valid when *class* is defined.

4.9.31 ExpireAfter

If you change the value of this parameter, it should be one of the first things you do in your configuration script.

This parameter controls the global value of the ExpireAfter parameter. See ⟨undefined⟩ [Spamming and security], page ⟨undefined⟩. This parameter controls the maximum time in minutes which a cfengine action is allowed to live. After this time a second cfengine agent will try to kill the cfengine which seems to have hung and attempt to restart the action. This is different from a TimeOut, where an internal alarm interrupt is used.

```
ExpireAfter = ( time-in-minutes )
```

This parameter may also be set per action in the action sequence by appending a pseudo-class called `ExpireAfter`*`time`*. For instance,

```
actionsequence = ( copy.ExpireAfter15 )
```

sets the expiry time parameter to 15 minutes for this copy command. This method should be considered old and deprecated however. As of version 2.1.0, you can define the expiry time on a per-command basis, as options of the form `expireafter=10`.

4.9.32 FullEncryption

From version 2.1.22. This is false for protocol compatability with older servers.

If this is set to true, directory filenames are encrypted in addition to file contents. If transferred data are not especially secret or travel over public networks this is a waste of CPU time. In high security environments it provides full privacy of data transmission.

```
FullEncryption = ( true )
```

4.9.33 HomePattern

```
HomePattern  = ( list of patterns )
```

The `homepattern` variable is used by the cfengine model for mounting nfs filesystems. See ⟨undefined⟩ [NFS resources], page ⟨undefined⟩. It is also used in the evaluation of the pseudo variable `home`, See Section 4.17 [files], page 86, Section 4.37 [tidy], page 136.

`homepattern` is in fact a list and is used like a wildcard or *pattern* to determine which filesystems in the list of mountables are home directories. See Section 4.28 [mountables], page 117. This relies on your sticking to a rigid naming convention as described in the first reference above.

For example, you might wish to mount (or locate directly if you are not using a separate partition for home directories) your home directories under `mountpattern` in directories `u1`, `u2` and so on. In this case you would define `homepattern` to match these numbers:

```
homepattern = ( u? )
```

Cfengine now regards any directory matching `$(mountpattern)/u?` as being a user login directory.

Suppose you want to create mount home directories under `$(mountpattern)/home` and make subdirectories for staff and students. Then you would be tempted to write:

```
HomePattern = ( home/staff home/students )
```

Unfortunately this is not presently possible. (This is, in principle, a bug which should be fixed in the future.) What you can do instead is to achieve the same this as follows:

```
MountPattern = ( /$(site)/$(host) /$(site)/$(host)/home )
HomePattern = ( staff students )
```

4.9.34 HostnameKeys

If this variable is set to true/on, it causes cfagent to lookup and store trusted public keys according to their DNS fully qualified host name, instead of using the IP address. This can be useful in environments where hosts do not have fixed IP addresses, but do have fixed hostnames.

```
HostnameKeys = ( on )
```

This method of storing keys is not recommended for sites with fixed IP addresses, since it removes one security barrier from a potential attacker by potentially allowing DNS spoofing.

Note that there is a corresponding variable to be set in '`cfrun.hosts`' which must be set for consistency.

4.9.35 IfElapsed

If you change the value of this parameter, it should be one of the first things you do in your configuration script.

This parameter controls the global value of the IfElapsed parameter, See ⟨undefined⟩ [Spamming and security], page ⟨undefined⟩. This parameter controls the minimum time which must have elapsed for an action in the action sequence before which it will be executed again.

```
IfElapsed = ( time-in-minutes )
```

This parameter may also be set per action in the action sequence by appending a pseudo-class called `IfElapsed`*`time`*. For instance,

```
ActionSequence = ( copy.IfElapsed15 )
```

sets the elapsed time parameter to 15 minutes for this copy command. This method should be considered old and deprecated however. As of version 2.1.0, you can define the expiry time on a per-command basis, as options of the form `ifelapsed=15`.

4.9.36 Inform

```
Inform = ( on/off )
```

This variable switches on the output level whereby cfengine reports changes it makes during a run. Normally only urgent messages or clear errors are printed. Setting `Inform` to `on` makes cfengine report on all actions not explicitly cancelled with a 'silent' option. To set this output level one writes:

```
classes::

  Inform = ( on )
```

4.9.37 InterfaceName

If you have an operating system which is installed on some non-standard hardware, you might have to specifically set the name of the network interface. For example:

```
control:

  nextstep.some::

     InterfaceName = ( en0 )

  nextstep.others::

     InterfaceName = ( ec0 )
```

It is only necessary to set the interface name in this fashion if you have an operating system which is running on special hardware. Most users will not need this. The choice set here overrides the system defaults and the choices made in the '`cfrc`' file, See Section 7.2 [cfrc resource file], page 154.

4.9.38 FileExtensions

This list may be used to define a number of extensions which are regarded as being plain files by the system. As part of the general security checking cfengine will warn about any directories which have names using these extensions. They may be used to conceal directories.

```
FileExtensions = ( c o gif jpg html )
```

4.9.39 LastSeen

This option is true by default. If set to off or false it prevents cfengine and/or cfservd from learning about last times hosts were observed connecting to one another. Some users with broken resolvers (particularly in view of the change over to IPv6 compatible libraries) might find this useful when processes appear to hang on connecting.

```
LastSeen = ( off )
```

4.9.40 LastSeenExpireAfter

This value (in days) sets the time after which unseen friend hosts are purged from the 'last seen' database, as viewed by the `FriendStatus` function, See Section 4.6 [alerts], page 28.

```
LastSeenExpireAfter = ( 2 )
```

4.9.41 LinkCopies

This list is used to define a global list of names or patterns which are to be linked symbolically rather than copied. For example

```
excludelinks = ( *.gif *.jpg )
```

The same facility can be specified for each individual link operation using the `symlink` option See Section 4.11 [copy], page 58.

Note that all entries defined under a specified class are valid only as long as that class is defined. For instance

```
class::

   LinkCopies = ( pattern )
```

would define a pattern which was only valid when *class* is defined.

4.9.42 LogDirectory

This is now deprecated.

Specify an alternative directory for keeping cfengine's log data. This defaults to '`/var/run/cfengine`' or '`/var/cfengine`'.

```
LogDirectory = ( /var/cfengine )
```

4.9.43 LogTidyHomeFiles

```
LogTidyHomeFiles = ( off )
```

If set to "off", no log is made of user files, in their home directories, of the files which are tidied by cfengine.

4.9.44 moduledirectory

```
moduledirectory  = ( directory for plugin modules )
```

This is the directory where cfengine will look for plug-in modules for the actionsequence, See Section 4.9.3 [actionsequence], page 34. Plugin modules may be used to activate classes using special algorithms. See ⟨undefined⟩ [Writing plugin modules], page ⟨undefined⟩. This variable defaults to '`/var/cfengine/modules`' for privileged users and to '`$HOME)/.cfengine/modules`' for non-privileged users.

4.9.45 mountpattern

`mountpattern  = ( mount-point )`

The `mountpattern` list is used by the cfengine model for mounting nfs filesystems. See ⟨undefined⟩ [NFS resources], page ⟨undefined⟩. It is also used in the evaluation of the pseudo variable `home`, See Section 4.17 [files], page 86, Section 4.37 [tidy], page 136.

It is used together with the value of `homepattern` to locate and identify what filesystems are local to a given host and which are mounted over the network. For this list to make sense you need to stick to a rigid convention for mounting your filesystems under a single naming scheme as described in the section mentioned above. If you follow the recommended naming scheme then you will want to set the value of mountpattern to

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

which implies that cfengine will look for local disk partitions under a unique directory given by the name of the host and site. Any filesystems which are physically located on the current host lie in this directory. All mounted filesystems should lie elsewhere. If you insist on keeping mounted file systems in more than one location, you can make a list like this:

```
mountpattern = ( /$(site)/users /$(site)/projects )
```

4.9.46 netmask

`netmask = ( aaa.bbb.ccc.ddd )`

The netmask variable defines the partitioning of the subnet addresses on your network. Its value is defined by your network administrator. On most systems it is likely to be `255.255.255.0`. This is used to configure the network interface in `netconfig`. See Section 4.9.3 [actionsequence], page 34.

Every host on the internet has its own unique address. The addresses are assigned hierarchically. Each network gets a *domain name* and can attach something like 65,000 hosts to that network. Since this is usually too many to handle in one go, every such network may be divided up into subnets. The administrator of the network can decide how the division into subnets is made. The decision is a trade-off between having many subnets with few hosts, or many hosts on few subnets. This choice is made by setting the value of a variable called `netmask`. The netmask looks like an internet address. It takes the form:

```
aaa.bbb.ccc.mmm
```

The first two numbers '`aaa.bbb`' are the address of the domain. The remainder '`ccc.mmm`' specifies both the subnet and the hostname. The value of `netmask` tells all hosts on the network: how many of the bits in the second half label different subnets and how many label different hosts on each of the subnets?

The most common value for the netmask is '`255.255.255.0`'. It is most helpful to think of the netmask in terms of bits. Each base-10 number between 0-255 represents 8 bits which are either set or not set. Every bit which is set is a network address and every bit which is zero is part of a host address. The first two parts of the address '`255.255`' always takes these values. If the third number is '`255`', it means that the domain is divided up into 256 sub networks and then the remaining bits which are zero can be used to give 255 different host addresses on each of the subnets.

If the value had been '255.255.255.254', the network would be divided up into 2^15 subnets, since fifteen of the sixteen bits are one. The remaining bit leaves enough room for two addresses 0 and 1. One of those is reserved for *broadcasts* to all hosts, the other can be an actual host — there would only be room for one host per subnet. This is a stupid example of course, the main point with the subnet mask is that it can be used to trade subnets for hosts per subnet. A value of '255.255.254.0' would allow 128 different subnets with $2 * 256 - 1 = 511$ hosts on each.

We needn't be concerned with the details of the netmask here. Suffice it to say that its value is determined for your entire domain by the network administrator and each host has to be told what the value is.

Each host must also know what convention is used for the *broadcast address*. This is an address which hosts can send to if they wish to send a message to every other host on their subnet simultaneously. It is used a lot by services like NIS to ask if any hosts are willing to perform a particular service. There are two main conventions for the broadcast address: address zero (all host bits are zero) and the highest address on the subnet (all host bits are ones). The convention can be different on every subnet and it is decided by the network administrator. When you write a cfengine program you just specify the convention used on your subnet and cfengine works out the value of the broadcast address from the netmask and the host address See Section 4.8 [broadcast], page 32. Cfengine works out the value of the broadcast address using the value of the netmask.

4.9.47 NonAlphaNumFiles

If enabled, this option causes cfengine to detect and disable files which have purely non-alphanumeric filenames, i.e. files which might be accidental or deliberately concealed. The files are then marked with a suffix `.cf-nonalpha` and are rendered visible.

```
NonAlphaNumFiles = ( on )
```

These files can then be tidied by searching for the suffix. Note that alphanumeric means ascii codes less than 32 and greater than 126.

4.9.48 nfstype

```
nfstype = ( nfs-type )
```

This variable is included only for future expansion. If you do not define this variable, its value defaults to "nfs".

At present cfengine operates only with NFS (the network file system). When cfengine looks for network file systems to mount, it adds lines in the filesystem table ('`/etc/fstab`','`/etc/checklist`' etc.) to try to mount filesystems of type "nfs". In principle you might want to use a completely different system for mounting filesystems over the network, in which case the 'mount type' would not be "nfs" but something else.

At the time of writing certain institutions are replacing NFS with AFS (the Andrew filesystem) and DFS (from the distributed computing environment). The use of these filesystems really excludes the need to use the mount protocol at all. In other words if you are using AFS or DFS, you don't need to use cfengine's mounting commands at all.

4.9.49 PortageInstallCommand

Sets the path to 'emerge' which is used to install packages that need to be installed under Gentoo Portage.

```
PortageInstallCommand = ( "/usr/bin/emerge --nocolor %s" )
```

By default, this variable is not set, meaning that any packages with action=install will NOT be installed if installation is required. Note the quotes around the string, and the `%s` is replaced with the list of packages to be installed, each separated by a space.

4.9.50 RepChar

```
RepChar = ( character )
```

The value of this variable determines the characters which is used by cfengine in creating the unique filenames in the file repository. Normally, its value is set to '`_`' and each '`/`' in the path name of the file is changed to '`_`' and stored in the repository. If you prefer a different character, define it here. Note that the character can be quoted with either single or double quotes in order to encompass spaces etc.

4.9.51 Repository

```
Repository  = ( directory )
```

Defines a special directory where all backup and junk files are collected. Files are assigned a unique filename which identifies the path from which they originate. This affects files saved using `disable`, `copy`, `links` and `editfiles` See ⟨undefined⟩ [Disabling and the file repository], page ⟨undefined⟩.

4.9.52 RPMcommand

The default value of the Red Hat Package manager command '`/bin/rpm`' can be altered for non-standard systems with this variable.

```
RPMcommand = ( /usr/bin/rpm )
```

4.9.53 RPMInstallCommand

Sets the command used to install packages that need to be installed under the RPM package manager.

```
RPMInstallCommand = ( "/usr/bin/pkgmgr %s" )
```

By default, this variable is not set, meaning that any packages with action=install will NOT be installed if installation is required. Note the `"`'s around the string, and the %s is replaced with the list of packages to be installed, each separated by a ' ' (space).

4.9.54 Schedule

```
schedule  = (  Min00_05 Min30_35 time class )
```

When `cfexecd` is used in daemon mode, it defaults to running once an hour, on the hour, i.e..

```
schedule  = (  Min00_05 )
```

This can be extended to make the agent run more often. The time specifiers are cfengine classes, and are written as intervals of time rather the precise times. Cfengine's time resolution is purposely limited to five minutes because the auto-correlation time of user resources is generally greater than this. Thus, it is assumed that precision timing is not required and the start time of cfengine, when scheduled in daemon mode, is not better than a few minutes. The daemon does not require precision, but offers many other strategic features for load balancing and security.

Other time classes can be used in the schedule list, but note that cfexecd will not run the agent more than once every five minutes. This is treated as a fundamental granularity.

4.9.55 SecureInput

```
SecureInput = ( on )
```

If this is set cfengine will not import files which are not owned by the uid running the program, or which are writable by groups or others.

4.9.56 SensibleCount

```
SensibleCount  = ( count )
```

This variable is used by the action `required`. It defines for cfengine what you consider to be the minimum number of files in a 'required' directory. If you declare a directory as being required, cfengine will check to see if it exists. Then, if the directory contains fewer than the value of `sensiblecount` files, a warning is issued. The default value for this variable is 2.

4.9.57 SensibleSize

```
SensibleSize  = ( size )
```

This variable is used by the action `required`. It defines for cfengine what you consider to be the minimum size for a 'required' file. If you declare a file as being required, cfengine will check to see if the file exists. Of course, the file may exist but be empty, so the size of the file is also checked against this constant. If the file is smaller than the value of `sensiblesize` a warning is issued. The default value for this variable is 1000 bytes.

4.9.58 ShowActions

```
ShowActions  = ( on )
```

This causes cfengine to produce detailed output of what action is being carried out as part of the prefix information during output. This is intended only for third party tools which collect and parse the cfengine output. It will be of little interest to humans.

4.9.59 SingleCopy

```
singlecopy = ( path_and_filename_wildcard )
```

If a `singlecopy` pattern is defined the behavior of copy: is modified so that a given destination file, matching the pattern, will only be updated once. In other words, if someone tries to copy more then one source file to the same location, the destination will not be overwritten in the same run. If the path name and wildcard is any of '*', 'on' or 'true', then the list applies to all files. For example:

```
control:

  actionsequence = ( copy )
```

```
  singlecopy = ( /tmp/* )

  addinstallables = ( zzz )

 zzz::

  autodefine = ( /tmp/* )

copy:

 /etc/passwd dest=/tmp/destination type=binary
 /etc/group  dest=/tmp/destination type=binary

alerts:

 zzz::

  "Copied something in /tmp"
```

Note (Warning) that this feature has several problems. It assumes an order dependence that cfengine generally tries to avoid. The first copy that takes place wins. Also, if files are locked at different times, this can result in oscillations between several different source files. e.g.

```
copy:

 /etc/passwd dest=/tmp/bla type=binary ifelapsed=2
 /etc/group  dest=/tmp/bla type=binary ifelapsed=1
```

In order to avoid explicit looping, cfengine assumes that a file has been copied even if no actual copy took place – i.e. as long as a file is apparently up to date, that counts as a valid copy update and the promise/action is considered done. If this were not the case, then the following promises would still be in line for execution and cfengine would loop between the different versions on subsequent invocations.

4.9.60 site/faculty

```
site    = ( sitename )
faculty = ( facultyname )
```

This variable defines a convenient name for your site configuration. It is useful for making generic rules later on, because it means for instance that you can define the name of a directory to be

```
/$(site)/$(host)/local
```

without having to redefine the rule for a specific site. This is a handy trick for making generic rules in your files which can be imported into a configuration for any site.

`faculty` is a synonym for `site`. The two names may be used interchangeably.

4.9.61 SkipIdentify

```
SkipIdentify = ( true )
```

This is the client side directive corresponding to the server directive `SkipVerify`. It tells cfengine not to assume that the client is registered in the Domain Name Service (DNS).

Sometimes the assumption of DNS registration can break connectivity between hosts, particularly if firewalls or Network Address Translation is in use.

4.9.62 smtpserver

```
smtpserver = ( mailhost )
```

This variable specified the destination for Email sent by cfexecd.

4.9.63 SplayTime

```
SplayTime = ( time-in-minutes )
```

This variable is used to set the maximum time over which cfengine will share its load on a server, See ⟨undefined⟩ [Splaying host times], page ⟨undefined⟩.

4.9.64 Split

```
Split  = ( character )
```

The value of this variable is used to define the list separator in variables which are expected to be treated as lists. The default value of this variable is the colon '`:`'. Cfengine treats variables containing this character as lists to be broken up and iterated over, See Section 4.2.3 [Iteration over lists], page 18.

This typically allows communication with PATH-like environment variables in the shell.

4.9.65 SpoolDirectories

A list of additional spool directories for cfengine to police. In these directories, filenames should correspond to existing users of the system. When users lost their accounts, this list plus the mail spool directory will be checked for files owned by deprecated users. See also: `DeleteNonOwnerFiles`, `DeleteNonUserFiles`.

```
SpoolDirectories = ( /var/spool/cron/crontabs /var/spool/cron/atjobs )
```

4.9.66 SUNInstallCommand

Sets the command used to install packages that need to be installed under the SUN package manager.

```
SUNInstallCommand = ( "/usr/bin/pkgmgr %s" )
```

By default, this variable is not set, meaning that any packages with action=install will NOT be installed if installation is required. Note the `"`'s around the string, and the %s is replaced with the list of packages to be installed, each separated by a ' ' (space).

4.9.67 suspiciousnames

```
SuspiciousNames = ( .mo lrk3 )
```

Filenames in this list are treated as suspicious and generate a warning as cfengine scans directories. This might be used to detect hacked systems or concealed programs. Checks are only made in directories which cfengine scans in connection with a command such as files, tidy or copy.

4.9.68 sysadm

```
sysadm = ( mail address )
```

The mail address of your system administrator should be placed here. This is used in two instances. If cfengine is invoked with the option `-a`, then it simply prints out this value. This is a handy feature for making scripts.

The administrators mail address is also written into the personal log files which cfengine creates for each user after tidying files, so you should make this an address which users can mail if they have troubles.

4.9.69 Syslog

```
Syslog = ( on/off )
```

This variable activates syslog logging of cfengine output at the 'inform' level.

To set this output level one writes:

```
classes::

  Syslog = ( on )
```

4.9.70 SyslogFacility

```
SyslogFacility = ( facility )
```

This variable alters the syslog facility level. e.g.

```
SyslogFacility = ( LOG_LOCAL1 )
```

Valid arguments are

```
LOG_USER
LOG_DAEMON
LOG_LOCAL0
LOG_LOCAL1
LOG_LOCAL2
LOG_LOCAL3
LOG_LOCAL4
```

4.9.71 timezone

```
timezone = ( 3-character timezone )
```

The timezone variable is a list of character strings which define your local timezone. Normally you will only need a single timezone, but sometimes there are several aliases for a given timezone e.g. MET and CET are synonymous. Currently only the first three characters of this string are checked against the timezone which cfengine manages to glean from the system. If a mismatch is detected a warning message is printed. cfengine does not attempt to configure the timezone. This feature works only as a reminder, since the timezone should really be set once and for all at the time the system is installed. On some systems you can set the timezone by editing a file, a procedure which you can automate with cfengine See Section 4.16 [editfiles], page 73.

The value of the `timezone` can be accessed by variable substitution in the usual way. It expands to the first item in your list.

```
shellcommands:

      "echo ${timezone} | mail ${sysadm}"
```

4.9.72 TimeOut

```
TimeOut = ( 10 )
```

The default timeout for network connections is 10 seconds. This is too short on some routed networks. It is not permitted to set this variable smaller than 3 seconds or larger than 60 seconds. A timeout is generated by an 'alarm' interrupt within an executing agent. This is constrasted with ExpireAfter, in which a second agent is required to interrupt the activity.

4.9.73 Verbose

```
Verbose = ( on/off )
```

This variable switches on the output level whereby cfengine reports everything it does during a run in great detail. Normally only urgent messages or clear errors are printed, See Section 4.9.36 [Inform], page 46. This option is almost equivalent to using the `--verbose` of `-v` command-line options. The only difference is that system environment reporting information, which is printed prior to parsing, is not shown. To set this output level on selected hosts one writes:

```
classes::

  Verbose = ( on )
```

For related more limited output, See Section 4.9.36 [Inform], page 46.

4.9.74 Warnings

```
Warnings = ( on/off )
```

This variable switches on the parser-output level whereby cfengine reports non-fatal warnings. This is equivalent to setting the command line switch `--no-warn`, or `-w`. To set this output level on selected hosts one writes:

```
classes::

  Warnings = ( on )
```

4.9.75 WarnNonUserFiles

If this parameter is set to true, cfengine will warn about files in spool directories which do not have a name belonging to a known user id.

See also `DeleteNonUserFiles`.

4.9.76 WarnNonOwnerFiles

If this parameter is set to true, cfengine will warn about files on mailservers whose names do not correspond to a known user name, but might be owned by a known user.

```
SpoolDirectories = ( /var/spool/cron/crontabs )

WarnNonOwnerFiles = ( true )
```

See also `DeleteNonOwnerFiles`. This generalizes and succeeds `DeleteNonOwnerMail`.

4.9.77 WarnNonUserMail

If this parameter is set to true, cfengine will warn about mail files on mailservers which do not have a name belonging to a known user id. This does not include lock files.

4.9.78 WarnNonOwnerMail

If this parameter is set to true, cfengine will warn about files on mailservers whose names do not correspond to a known user name, but might be owned by a known user.

4.10 classes

The `classes` keyword is an alias for `groups` as of version 1.4.0 of cfengine.

4.11 copy

Cfengine copies files between locally mounted filesystems and via the network from registered servers. The copy algorithm avoids race-conditions which can occur due to network and system latencies by copying first to a file called '*file*.cfnew' on the local filesystem, and then renaming this quickly into place. The aim of this roundabout procedure is to avoid situations where the direct rewriting of a file is interrupted midway, leaving a partially written file to be read by other processes. Cfengine attempts to preserve hard links to non-directory file-objects, but see the caution below.

Caution should be exercised in copying files which change rapidly in size. This can lead to file corruption, if the size changes during copying. Cfengine attempts to prevent this during remote copies.

The syntax summary is:

```
copy:

   class::

      master-file
                        dest=destination-file
                        mode=mode
                        owner=owner
                        group=group
                        action=warn/silent/fix
                        backup=true/false/timestamp
                        repository=backup directory
                        stealth=true/on/false/off
                        timestamps=preserve/keep
                        symlink=pattern

                        include=pattern
                        exclude=pattern
                        ignore=pattern
                        filter=filteralias
                        xdev=true/on/false/off

                        recurse=number/inf/0
                        type=ctime/mtime/checksum/sum/byte/binary/any
                        linktype=absolute/symbolic/relative/hard/none/copy
                        typecheck=true/on/false/off
                        define=class-list(,:.)
                        elsedefine=class-list(,:.)

                        force=true/on/false/off
                        forcedirs=true/on/false/off
                        forceipv4=true/on/false/off
                        size=size limits
                        server=server-host
                        failover=classes

                        trustkey=true/false
                        secure=[deprecated]
                        encrypt=true/false
                        verify=true/false
                        oldserver=true/false

                        purge=true/false

                        syslog=true/on/false/off
                        inform=true/on/false/off

                        findertype=MacOSX finder type
```

Note that the `$(this)` variable may be used in the source and destination strings in order to tailor the filename when downloading from multiple sources (see below).

`dest` The destination file is the only obligatory item. This must be the name of an object which matches the type of the master object i.e. if the master is a plain file, the destination must also be the explicit name of a plain file. An implicit 'copy file to directory' syntax is not allowed. Symbolic links are copied as symbolic links, plain files are copied as plain files and special files are copied as special files. The `recurse` option is required to copy the contents of subdirectories.

If the destination file name is of the form '`filename/..namedfork/rsrc`', then it is assumed that you are copying the resource fork of a file to an HFS+ file system on OS X Jaguar. In the absence of the destination file being in this form (just dest=filename), cfengine will assume that you are working with the data fork of the file.

For a resource fork copy to properly work, the data fork must have already been copied. Ie the OS will not allow you to copy the resource fork for a file that does not exist. And, copying a data fork after the resource fork will overwrite the resource fork. So, order is important. Copy the data fork, first. Then, copy the resource fork.

To split the data and resource forks of a file into two parts, open up a terminal. The following commands will copy MyFile 's data and resource forks into two separate files which can then be recombined by cfengine:

```
cp MyFile MyFile-datafork
cp MyFile/..namedfork/rsrc MyFile-rsrcfork
```

`mode, owner, group`
: The file mode, owner and group of the images are specified as in the `files` function See Section 4.17 [files], page 86.

`action`
: The action may take the values `warn`, `silent` or `fix`. The default action is `fix`, i.e. copy files. If `warn` is specified, only a warning is issued about files which require updating. If `silent` is given, then cfengine will copy the files but not report the fact.

`force`
: If set to 'true', this option causes cfengine to copy files regardless of whether it is up to date.

`forceipv4`
: If you are working on an ipv6 enabled pair of hosts, cfengine will normally select ipv6 for communication between them. If you wish to force the use of ipv4 for some reason, set this option to true.

`forcedirs`
: If set to 'true', this option causes files or links which block the creation of directories, during recursive copying, to be moved aside forcably. A single non-supressable warning is given when this occurs; the file is moved to filename'`.cf-moved`'.

`backup`
: If the `backup` option is set to "false", cfengine will not make a backup copy of the file before copying. The default value is "true". If the option "timestamp" is chosen, a unique timestamp will be appended to the saved filename.

`repository`
: This allows a local override of the `Repository` variable, on an item by item basis. If set to "off" or "none" it cancels the value of a global repository.

  Copy makes a literal image of the master file at the destination, checking whether the master is newer than the image. If the image needs updating it is copied. Existing files are saved by appending `.cfsaved` to the filename.

`stealth` If set to 'on' causes cfengine to preserve atime and mtime on source files during *local* file copies. File times cannot be preserved on remote copies. This option should normally only be used together with a checksum copy, since preserving atime and mtime implies changing ctime which will force continual copying. This is a weakness in the Unix file system. Ctime cannot be preserved. Before version 1.5.0, there was a typo which made this option active on many file copies.

`timestamps`
If this is set to 'preserve' or 'keep', the times of the source files are kept by the destination files during copying. This is like the 'p' option of the `tar` command.

`recurse` Specifies the depth of recursion when copying whole file-trees recursively. The value may be a number or the keyword `inf`. Cfengine crosses device boundaries or mounted filesystems when descending recursively through file trees. To prevent this it is simplest to specify a maximum level of recursion.

`symlink` This option may be repeated a number of times to specify the names of files, or wildcards which match files which are to be symbolically linked instead of copied. A global list of patterns can also be defined in the control section of the program See Section 4.9.41 [linkcopies], page 47.

`ignore` This works like the global ignore directive but here you may provide a private list of ignorable directories and files. Unlike include, exclude this affects the way cfengine parses directory trees.

`include` This option may be repeated a number of times to specify the names of files, or wildcards which match files which are to be included in a copy operation. Specifying one of these automatically excludes everything else except further include patterns. A global list of patterns can also be defined in the control section of the program.

If the `purge` option is used in copying, then the `ignore` option has the effect of the excluding files from the purge, i.e. `ignore` means 'keep' the named files.

`exclude` This option may be repeated a number of times to specify the names of files, or wildcards which match files which are to be excluded from a copy operation. A global list of patterns can also be defined in the control section of the program 'excludes' override 'includes'. See Section 4.9.30 [excludelinks], page 43.

`xdev` Prevents cfengine from descending into file systems that are not on the same device as the root of the rescurion path.

`type` Normally cfengine uses the ctime date-stamps on files to determine whether a file needs to be copied: a file is only copied if the master is newer than the copy or if the copy doesn't exist. If the type is set to '`checksum`' or '`sum`', then a secure MD5 checksum is used to determine whether the source and destination files are identical. If '`byte`' or '`binary`' is specified, a byte by byte comparison is initiated. An '`mtime`' comparison does not take into account changes of file permissions, only modifications to the contents of the files.

`findertype`
: Sets the four letter file type code in an HFS+ file system on Mac OS X Jaguar. For example, the four letter code APPL indicates the file is an Application (and will be executed when double-clicked). The four letter code TEXT indicates the file is a text file and will be opened by the default text editor.

  If the file also has an extension (for example `.txt`), then if setting the finder type code, you should make sure your finder type code does not conflict with the file extension.

  Files both without extensions and finder type codes are mostly useless to OS X, so be sure to do one or the other!

  Also note that finder type codes should not be applied to the resource forks of files.

`server`
: If you want to copy a file remotely from a server, you specify the name of the server here. This must be the name of a host which is running the `cfservd` daemon, and you must make sure that you have defined the variable `domain` in the control section of the '`cfagent.conf`' file. If you don't define a domain you will probably receive an error of the form 'cfengine: Hey! cannot stat file'. If the server name is 'localhost', cfengine will perform a local copy, without using a connection to cfservd.

`failover`
: If a file copy fails due to an error, the classes in this assignment will become active, allowing failover rules to become active.

`oldserver`
: If this is true, cfengine uses the old protocol specification for temporary compatibility with early version 2 alphas.

`trustkey`
: This option defaults to 'no' or 'false'. If set to true, cfagent will accept a public key from a server whose public key is presently unknown to the agent, on trust. This option should be used to bootstrap public key transfer between hosts. Once a public key has been accepted, it will not be replaced automatically. Dated public keys must be removed by hand.

`encrypt`
: Has an effect only when used in conjuction with copy from a remote file server. This causes cfengine to use encryption and one-time keys on transferred data. (This requires RSA keys to be installed on both client and server hosts, and provides strong authentication and encryption, using random session keys.) The preferred algorithm is Blowfish, with a 128 bit key. Generally speaking the only case in which this function makes sense is in transferring shadow password files. Encrypting the transfer of system binaries makes little sense. Note: the encryption keys required to get files from cfservd are those for the user under which cfservd is running (normally root).

`verify`
: If verify is true, cfagent attempts to verify the integrity of a remote file transfer before the new file is installed. This takes time, since an MD5 computation and transaction must take place.

`size`
: With this option you can specify that a file is only to be copied if the source file meets a size critereon. This could be used to avoid installing a corrupted

file (the copying of an empty password file, for instance). Sizes are in bytes by default, but may also be quoted in kilobytes or megabytes using the notation:

```
numberbytes
numberkbytes
numbermbytes
```

Only the first characters of these strings are significant, so they may be written however is convenient: e.g. *14kB*, *14k*, *14kilobytes* etc. Examples are:

```
size=<400  # copy if file size is < 400 bytes
size=400   # copy if file size is equal to 400 bytes
size=>400  # copy if file size > 400 bytes
```

`linktype` This option determines the type of link used to make links. This only applies if the file is linked rather than copied because it matches a pattern set by `symlink`. The default type is a direct symbolic link. The values '`relative`' or '`absolute`' may be used, but hard links may not be created in place of copied files, since hard links must normally reside on the same filesystem as their files, and it is assumed that most links will be between filesystems. If this value is set to `copy` or `none`, symbolic links will be replaced by actual copies of the files they point to. Note that for directories, this option is ignored.

`typecheck`
Controls whether cfengine allows files of one type to overwrite files of another type, i.e. switches on/off errors if source and existing destination files do not match in type, e.g. if a file would overwrite a directory or link. The default is on for safety reasons.

`define` This option is followed by a list of classes which are to be 'switched on' if and only if the named file was copied. In multiple (recursive) copy operations the classes become defined if any of the files in the file tree were copied. This feature is useful for switching on other actions which are to be performed after the installation of key files (e.g. package installation scripts etc).

`purge` If this option is set to true, cfengine will remove files in the destination directory which are not also in the source directory. This allows exact images of filesystems to be mantained. Note that if the copy command has includes or excludes or ignored files, cfengine will purge only those files on the client machine which are also on the server. Included files are not purged. This means that some files (such as system specific work files) can be excluded from copies without them being destroyed. Note that purging is disallowed if contant with a remote server fails. This means that local files will not be destroyed by a denial of service attack. *You should not use this option to synchronize NFS mounted file systems. If the NFS server goes down, cfengine cannot then tell the difference between a valid empty directory and a missing NFS file system. If you use purge, use a remote copy also.* If we specify purge, then the following options will also be set and cannot be altered: `forcedirs=true`, `typecheck=false`, since other defaults could be very destructive.

Example:

```
copy:

      /local/etc/aliases dest=/etc/aliases m=644 o=root g=other
      /local/backup-etc  dest=/etc

   solaris::

      /local/etc/nsswitch.conf dest=/etc/nsswitch.conf
```

In the first example, a global aliases file is copied from the master site file '`/local/etc/aliases`' to '`/etc/aliases`', setting the owner and protection as specified. The file gets installed if '`/etc/aliases`' doesn't exist and updated if '`/local/etc/aliases`' is newer than '`/etc/aliases`'. In the second example, '`backup-etc`' is a directory containing master configuration files (for instance, '`services`', '`aliases`', '`passwd`'...). Each of the files in '`backup-etc`' is installed or updated under '`/etc`'. Finally, a global '`nsswitch.conf`' file is kept up to date for Solaris systems.

The `home` directive can be used as a destination, in which case cfengine will copy files to every user on the system. This is handy for distributing setup files and keeping them updated:

```
copy:

   /local/masterfiles/.cshrc  dest=home/.cshrc mode=0600
```

You can force the copying of files, regardless of the date stamps by setting the option `force=true` or `force=on`. The default is `force=false` or `force=off`.

4.11.1 Hard links in copying

Hard links are not like symbolic links, they are not merely pointers to other files, but alternative names for the same file. The name of every file is a hard link, the first so to speak. You can add additional names which *really are* the file, they are not just pointers. For the technically minded, they are not separate inodes, they are additional directory references to the same inode. When you perform a copy operation on multiple files, cfengine attempts to preserve hard links but this is a difficult task.

Because a hard link just looks like an ordinary file (it cannot be distingiushed from the original, the way a symbolic link can) there is a danger that any copy operation will copy two hard links to the same file as two separate copies of the same file. The difference is that changes a hard-linked file propagate to the links, whereas two copies of a file are completely independent thereafter. In order to faithfully reproduce all hardlinks to all files, cfengine needs to examine every file on the same filesystem and check whether they have the same inode-number. This would be an enourmous overhead, so it is not done. Instead what happens is that cfengine keeps track of only the files which it is asked to examine, for each atomic copy-command, and makes a note of any repeated inodes within this restricted set. It does not try to go off, wandering around file systems looking to other files which might be hardlinks.

To summarize, cfengine preserves hardlinks during copying, only within the scope of the present search. No backups are made of hard links, only of the first link or name of the file is backed up. This is a necessary precaution to avoid dangling references in the inode table. As a general rule, hard links are to be avoided because they are difficult to keep track of.

4.11.2 Too many open files

In long recursive copies, where you descend into many levels of diretories, you can quickly run out of file descriptors. The number of file descriptors is a resource which you can often set in the shell. It is a good idea to set this limit to a large number on a host which will be copying a lot of files. For instance, in the C shell you would write,

```
limit descriptors 1024
```

Most systems should have adequate defaults for this parameter, but on some systems it appears to be set to a low value such as 64, which is not sufficient for large recursive tree searches.

4.12 defaultroute

Dynamical routing is not configurable in cfengine, but for machines with static routing tables it is useful to check that a default route is configured to point to the nearest gateway or router. The syntax for this statement is simply:

```
defaultroute:

   class::

      my_gateway
```

For example:

```
defaultroute:

  most::

     129.240.22.1

  rest::

     small_gw

  no_default_route::

     192.168.1.1
```

Gateways and routers usually have internet address `aaa.bbb.ccc.1` — i.e. the first address on the subnet. You may use the numerical form or a hostname for the gateway.

The class `no_default_route` is defined if the current host does not have a currently defined default route, but specifies one in its configuration.

4.13 disks

This is a synonyn for `required`, See Section 4.32 [required], page 129. This action tests for the existence of a file or filesystem. It should be called after all NFS filesystems have been mounted. You may use the special variable `$(binserver)` here.

```
disks:

  /filesystem freespace=size-limit define=class-list(,:.)

   inform=true
   log=true

   scanarrivals=true
   force=true

   ifelapsed=mins
   expireafter=mins
```

Files or filesystems which you consider to be essential to the operation of the system can be declared as 'required'. Cfengine will warn if such files are not found, or if they look funny.

Suppose you mount your filesystem `/usr/local` via NFS from some binary server. You might want to check that this filesystem is not empty! This might occur if the filesystem was actually *not* mounted as expected, but failed for some reason. It is therefore not enough to check whether the directory `/usr/local` exists, one must also check whether it contains anything sensible.

Cfengine uses two variables: `sensiblesize` and `sensiblecount` to figure out whether a file or filesystem is sensible or not. You can change the default values of these variables (which are 1000 and 2 respectively) in the `control` section. See Section 4.9 [control], page 33.

If a file is smaller than `sensiblesize` or does not exist, it fails the 'required' test. If a directory does not exist, or contains fewer than `sensiblecount` files, then it also fails the test and a warning is issued.

```
disks:

   any::

      /$(site)/$(binserver)/local
```

If you set the `freespace` variable to a value and set `inform=true`, cfagent issues warnings when free disk space falls below this threshold. Any define-classes also become defined in this instance. (the default units are kilobytes, but you may specify bytes or megabytes), e.g.

If the option `force=true` is used, cfengine will parse filesystems even on NFS mounted filesystems. Normally it does not make sense to check filesystems that are not native to the local host, but occasionally ne would like to force such a check in order to set a class, based on the result, for instance.

If the `scanarrivals` option is set, the agent will recursively descend through the file system building a database of file modification times. This data is used for research purposes and will eventually be used to trigger classes that determine optimal times for backup of filesystem.

4.14 directories

Directories declarations consist of a number of directories to be created. Directories and files may also be checked and created using the `touch` option in the `files` actions. See Section 4.17 [files], page 86.

The form of a declaration is:

```
directories:

   classes::

       /directory
                       mode=mode
                       owner=uid
                       group=gid
                       define=classlist
                       elsedefine=classlist
                       syslog=true/on/false/off
                       inform=true/on/false/off

                       ifelapsed=mins
                       expireafter=mins
```

For example

```
directories:

  class::

     /usr/local/bin  mode=755 owner=root group=wheel
```

The form of the command is similar to that of `files` but this command is only used to create new directories. Valid options are `mode`, `owner`, `group` and are described under `files` See Section 4.17 [files], page 86. This interface is only for convenience. It is strictly a part of the 'files' functionality and is performed together with other 'files' actions at run time.

The creation of a path will fail if one of the links in the path is a plain file or device node. A list of classes may optionally be defined here if a directory is created.

If the `owner` value is set to the literal "LastNode", then the owner will be exchanged for the last node of the path. This allows the creation of home directories owned by users.

```
control:

  homedirs = ( mark:simen:luke:aeleen )
```

```
directories:

  /home/$(listcontent) owner=LastNode
```

4.15 disable

Disabling a file means renaming it so that it becomes harmless. This feature is useful if you want to prevent certain dangerous files from being around, but you don't want to delete them— a deleted file cannot be examined later. The syntax is

```
disable:

  class::

     /filename
                      dest=filename

                      type=plain/file/link/links
                      rotate=empty/truncate/numerical-value
                      size=numerical-value
                      define=classlist
                      syslog=true/on/false/off
                      inform=true/on/false/off
                      repository=destination directory
                      action=disable/warn

                      ifelapsed=mins
                      expireafter=mins
```

If a destination filename is specified, cfagent renames the source file to the destination, where possible (renaming across filesystems is not allowed). If no destination is given, cfagent renames a given file by appending the name of the file with the suffix '`.cfdisabled`'. Note that directories are only renamed if they have a specific destination specified.

A typical example of a file you would probably want to disable would be the `/etc/hosts.equiv` file which is often found with the '`+`' symbol written in it, opening the system concerned to the entire NIS universe without password protection! Here is an example:

```
disable:

      /etc/hosts.equiv
      /etc/nologin
      /usr/lib/sendmail.fc

   sun4::

      /var/spool/cron/at.allow
```

Hint: The last example disables a file which restricts access to the `at` utility. Such a command could be followed by a file action, See Section 4.17 [files], page 86,

```
files:

   some::
```

```
        /var/spool/cron/at.allow =0644 N owner=root group=wheel touch
```

which would create an empty security file 'at.allow'. See also your system manual pages for the at command if you don't understand why this could be useful.

Disabling a link deletes the link. If you wish you may use the optional syntax

```
disable:

   /directory/name type=file
```

to specify that a file object should only be disabled if it is a plain file. The optional element type= can take the values plain, file, link or links. If one of these is specified, cfengine checks the type and only disables the object if there is a match. This allows you to disable a file and replace it by a link to another file for instance.

NOTE that if you regularly disable a file which then gets recreated by some process, the disabled file '*filename*.cfdisabled' will be overwritten each time cfengine disables the file and therefore the contents of the original are lost each time. The rotate facility was created for just this contingency.

The disable feature can be used to control the size of system log files, such as '/var/adm/messages' using a further option rotate. If the value rotate is set to 4, say,

```
 disable:

   filename  rotate=4
```

then cfengine renames the file concerned by appending '.1' to it and a new, empty file is created in its place with the same owner and permissions. The next time disable is executed '.1' is renamed to '.2' and the file is renamed '.1' and a new empty file is created with the same permissions. Cfengine continues to rotate the files like this keeping a maximum of four files. This is similar to the behaviour of syslog.

If you simply want to empty the contents of a log file, without retaining a copy then you can use rotate=empty or rotate=truncate. For instance, to keep control of your World Wide Web server logs:

```
disable:

   Sunday|Wednesday::

       /usr/local/httpd/logs/access_log  rotate=empty
```

This keeps a running log which is emptied each Sunday and Wednesday.

The size= option in disable allows you to carry out a disable operation only if the size of the file is less than, equal to or greater than some specified size. Sizes are in bytes by default, but may also be quoted in kilobytes or megabytes using the notation:

```
numberbytes
numberkbytes
numbermbytes
```

Only the first characters of these strings are significant, so they may be written however is convenient: e.g. *14kB*, *14k*, *14kilobytes* etc. Examples are:

```
size=<400  # disable if file size is < 400 bytes
size=400   # disable if file size is equal to 400 bytes
size=>400  # disable if file size > 400 bytes
```

This options works with `rotate` or normal disabling; it is just an extra condition which must be satisfied.

If a disable command results in action being taken by cfengine, an optional list of classes becomes can be switched on with the aid of a statement `define=`*`classlist`* in order to trigger knock-on actions.

The `repository` declaration allows a local override of the `Repository` variable, on an item by item basis. If set to "off" or "none" it cancels the value of a global repository and leaves the disabled file in the same directory.

4.16 editfiles

Performs ascii (line-based) editing on text-files or limited binary editing of files. If editing a file which has hard links to it, be aware that editing the file will destroy the hard link references. This is also the case with shell commands. You should avoid hard links whenever possible. The form of an editing command is `editfiles` can also search directories recursively through directories and edit all files matching a pattern, using `Include`, `Exclude`, and `Ignore` (see Recursive File Sweeps in the tutorial).

```
editfiles:

   class::

      { file-to-be-edited

      action "quoted-string..."
      }

      { directory-to-be-edited

      Recurse "inf"          # iterated over all files
      Filter  "filteralias"
      Include ".cshrc"
      Ignore  "bin"
      Ignore  ".netscape"
      action "quoted-string..."
      }
```

Here are some examples:

```
editfiles:

   sun4::

      { /etc/netmasks

      DeleteLinesContaining "255.255.254.0"
      AppendIfNoSuchLine "128.39  255.255.255.0"
      }

   PrintServers::
      { /etc/hosts.lpd

      AppendIfNoSuchLine "tor"
      AppendIfNoSuchLine "odin"
      AppendIfNoSuchLine "borg"
      }
```

The first of these affects the file '`/etc/netmasks`' on all SunOS 4 systems, deleting any lines containing the string "255.255.254.0" and Appending a single line to the file containing "128.39 255.255.255.0" if none exists already. The second affects only hosts in the class 'PrintServers' and adds the names of three hosts: tor, odin and borg to the file

'/etc/hosts.lpd' which specifies that they are allowed to connect to the printer services on any host in the class 'PrintServers'.

Note that single or double quotes may be used to enclose strings in cfengine. If you use single quotes, your strings may contain double quotes and vice-versa. Otherwise a double quoted string may not currently contain double quotes and likewise for single quoted strings.

As of version 2.0.6 quoted strings may contain escaped quotes using `\"`.

As of version 1.3.0, you can use the 'home' directive in edit filenames, enabling you to edit files for every user on the system, provided they exist. For example, to edit every user's login files, you would write

```
{ home/.cshrc

 AppendIfNoSuchLine "setenv PRINTER default-printer"
 AppendIfNoSuchLine "set path = ( $path /new/directory )"
}
```

If a user does not possess the named file, cfengine just skips that user. A new file is not created.

The meanings of the file-editing actions should be self-explanatory. Commands containing the word 'comment' are used to 'comment out' certain lines in a file rather than deleting them. `Hash` implies a shell comment of the type

```
# comment
```

`Slash` implies a comment of the C++ type:

```
// comment
```

`Percent` implies a comment of the type:

```
% comment
```

More general comment types may be defined using the `SetCommentStart`, `SetCommentEnd` and `CommentLinesMatching`, `CommentLinesStarting` functions.

A special group of editing commands is based on the POSIX Regular Expression package. These use regular expressions to search line by line through text and perform various editing functions. Searches are of two different types: "LineMatching" and "LineContaining". In the first case the regular expression must match the entire line exactly; in the latter, a substring is searched for in the file.

Some of these commands are based on the concept of a file pointer. The pointer starts at line one of the file and can be reset by 'locating' a certain line, or by using the reset-pointer commands. The current position of the pointer is used by commands such as `InsertLine` to allow a flexible way of editing the middle of files.

A simple decision mechanism is incorporated to allow certain editing actions to be excluded. For instance, to insert a number of lines in a file once only, you could write:

```
{ file

 LocateLineMatching "insert point..."
 IncrementPointer   "1"
```

```
 BeginGroupIfNoMatch "# cfengine - 2/Jan/95"
   IncrementPointer "-1"
   InsertLine "# cfengine - 2/Jan/95"
   InsertLine "/local/bin/start-xdm"
   DefineInGroup "AddedXDM"
 EndGroup
}
```

Since the first inserted line matches the predicate on subsequent calls, the grouped lines will only be carried out once. When the grouped lines are run, the '`AddedXDM`' class will be activated for use by a later part of the script.

The full list of editing actions is given below in alphabetical order. Note that some commands refer to regular expressions and some refer to 'literal strings' (i.e. any string which is not a regular expression). Variable substitution is performed on all strings. Be aware that symbols such as '`.`', '`*`' and so on are meta-characters in regular expressions and a backslash must be used to make them literal. The regular expression matching functions are POSIX extended regular expressions. See ⟨undefined⟩ [Regular expressions], page ⟨undefined⟩.

`AbortAtLineMatching` *quoted-regex*

This command sets the value of a regular expression. In all editing operations (except `FixEndOfLine` and `GotoLastLine`) which involve multiple replacements and searches, this expression marks a boundary beyond which cfengine will cease to look any further. In other words, if cfengine encounters a line matching this regular expression, it aborts the current action. BE CAREFUL with this feature: once set, the string remains set for the remainder of the current file. It might therefore interact in unsuspected ways with other search parameters. Editing actions are always aborted as soon as the abort expression is matched. Use `UnsetAbort` to unset the feature.

`Append` *quoted-string*

Add a line containing the quoted string to the end of the file. This should be used in conjunction with the decision structures `BeginGroupIfNoLineMatching` and `BreakIfLineMatches`.

`AppendIfNoLineMatching` *quoted-regex*`/ ''ThisLine''`

A new version of the older `AppendIfNoSuchLine` which uses a regular expression instead of a literal string. The line which gets appended must be set previously using `SetLine`. If `''ThisLine''` is given as the argument, the current value of then line buffer is assumed. This allows constructions for merging files on a convergent line-by-line basis:

```
editfiles:

 { /tmp/bla

 ForEachLineIn "/tmp/in"
    AppendIfNoLineMatching "ThisLine"
 EndLoop
 }
```

`AppendIfNoSuchLine` *quoted-string*
: Add a line containing the quoted string to the end of the file if the file doesn't contain the exact line already.

`AppendIfNoSuchLinesFromFile` *filename*
: For each line in the named file, call AppendIfNoSuchLine. This adds lines containing the strings listed in the named file to the end of the current file if the file doesn't contain the exact line already.

`AppendToLineIfNotContains` *quoted-string*
: This commands looks for an exact match of the quoted string in the current line. If the quoted string is not contained in the line, it is appended. This may be used for adding entries to a list.

`AutoCreate`
: If this command is listed anywhere in the file action list, cfengine will create the named file if it doesn't exist. Normally cfengine issues an error if the named file does not exist, but if this is set, notification of the file's absence is only in verbose output. New files are created with mode 644 (see also Umask), read access for everyone and write access for the cfengine user (normally root). Note that if you set this, BeginGroupIfFileIsNewer will always be true.

`AutomountDirectResources` *quoted-string*
: This command is designed to assist with automounter configuration for users wishing to use the automounter for NFS filesystems, but still use the cfengine mount model. Applied to the current file, it is equivalent to saying: for each of the mountable resources in the list See Section 4.28 [mountables], page 117, append if not found a line for a direct automount map command, to the current file. The string which follows can be used to specify any special mount options e.g. `''-nosuid''` for non setuid mounting (of all the mountables). Note that this is added to the current file and not to a file named '`/etc/auto_direct`'.

`Backup` *quoted-string*
: Set to true or false, on or off to set backup policy for this file. Default is on. The default is to produce time-stamped backups of files; this may be coded explicitly by setting to "timestamp" or "stamp". If set to "false" or "off", no backup is kept of the edited file. If the value is set to "single" or "one" then only the last version of the file is kept, overwriting any previously saved versions.

```
Backup "single"
```

`BeginGroupIfDefined` *quoted-string*
: The lines following, up to the first `EndGroup` are executed if the quoted class is defined. Edit groups may be nested.

`BeginGroupIfNotDefined` *quoted-string*
: The lines following, up to the first `EndGroup` are executed if the quoted class is not defined. Edit groups may be nested.

`BeginGroupIfFileExists` *quoted-string*
: The lines following, up to the first `EndGroup` are executed if the quoted filename exists (can be statted). Files which are not readable by the running process are for all intents and purposes non-existent. Edit groups may be nested.

`BeginGroupIfFileIsNewer` *quoted-string*
: The lines following, up to the first `EndGroup` are executed if the quoted filename is newer than the file being edited. Edit groups may be nested.

`BeginGroupIfLineContaining` *quoted-string*
: The lines following, up to the first `EndGroup` are executed if the quoted string appears in any line in the file. Edit groups may be nested.

`BeginGroupIfLineMatching` *quoted-regex*
: The lines following, up to the first `EndGroup` are executed if the quoted regular expression matches any line in the file. Edit groups may be nested.

`BeginGroupIfMatch` *quoted-regex*
: The lines following, up to the first `EndGroup` are executed if the quoted regular expression matches the current line. Edit groups may be nested.

`BeginGroupIfNoLineContaining` *quoted-string*
: The lines following, up to the first `EndGroup` are executed if the quoted string does not appear in any line in the file. Edit groups may be nested.

`BeginGroupIfNoLineMatching` *quoted-regex*
: The lines following, up to the first `EndGroup` are executed if the quoted regular expression does not match any line in the file. Edit groups may be nested.

`BeginGroupIfNoMatch` *quoted-regex*
: The lines following, up to the first `EndGroup` are executed if the quoted regular expression does not match the current line. Edit groups may be nested.

`BeginGroupIfNoSuchLine` *quoted-string*
: The lines following, up to the first `EndGroup` are executed if the quoted literal string does not match any line in the file. Edit groups may be nested.

`BreakIfLineMatches` *quoted-regex*
: Terminates further editing of the current file if the current line matches the quoted regular expression.

`CatchAbort`
: Edit actions which abort on failure (such as `LocateLineMatching`) will jump to the first instance of this marker instead of completely aborting an edit if this keyword occurs in an editing script. You can catch the exceptions thrown by the following commands: `CommentNLines`,`CommentToLineMatching`,`DeleteNLines`,`DeleteToLineMatching`, `HashCommentToLineMatching`,`IncrementPointer`, `LocateLineMatching`,`PercentCommentToLineMatching`, `RunScriptIf(No)LineMatching`,`UnCommentNLines`.

`CommentLinesMatching` *quoted-regex*
: Use the current value of the comment delimiters set using `SetCommentStart` and `SetCommentEnd` to comment out lines matching the given regular expression in quotes.

`CommentLinesStarting` *quoted-string*
: Use the current value of the comment delimiters set using `SetCommentStart` and `SetCommentEnd` to comment out lines starting with the quoted literal string.

`CommentNLines` *quoted-string*

Comments up to N lines from the current file, starting from the location of the current line pointer. If the end of the file is reached and less than N lines are deleted, a warning is issued, but editing continues. The current value of the comment delimiters is used to determine the method of commenting, (see `SetCommentStart`). After the operation the pointer points to the line after the commented lines.

`CommentToLineMatching` *quoted-regex*

Use the current value of the comment delimiters set using `SetCommentStart` and `SetCommentEnd` to comment out lines from the current position in a file to a line matching the given regular expression in quotes.

`DefineClasses "`*class1*`:`*class2*`:..."`

Activate the following colon, comma or dot-separated list of classes if and only if the file is edited.

`DefineInGroup "`*class1*`:`*class2*`:..."`

Activate the following colon, comma or dot-separated list of classes if the edit group is entered. This can be combined with other classes to identify what particular edits took place. Use DefineInGroup if you want to define a class or list of classes conditional on entry to a BeginGroup ... EngGroup block. For example,

```
editfiles:
    { /etc/inetd.conf
      BeginGroupIfNoSuchLine "$(myservice1)"
        Append "$(myservice1)"
DefineInGroup "myservice1_added"
      EndGroup

      BeginGroupIfNoSuchLine "$(myservice2)"
        Append "$(myservice2)"
DefineInGroup "myservice2_added"
      EndGroup
    }
```

This will define service_added and service_added_another_way if either line is added, but additionally myservice1_added if myservice1 was added and likewise for myservice2_added.

`DefineInGroup "`*class1*`:`*class2*`:..."`

Activate the following colon, comma or dot-separated list of classes if execution reaches the `BeginGroup` ... `EndGroup` section(s) containing this command. If you think you want to put DefineClasses within a `BeginGroup` ... `EndGroup` section, you actually want this.

`DeleteLinesAfterThisMatching` *quoted-regex*

Delete lines after the current position which match the quoted expression.

`DeleteLinesContaining` *quoted-string*`/DeleteLinesNotContaining` *quoted-string*

Delete all lines (not) containing the exact string quoted.

`DeleteLinesMatching` *quoted-regex*`/DeleteLinesNotMatching` *quoted-regex*

Delete all lines (not) fully matching the tied quoted regular expression.

`DeleteLinesStarting` *quoted-string*/`DeleteLinesNotStarting` *quoted-string*
: Delete all lines (not) beginning with the exact string quoted.

`DeleteLinesNotContainingFileItems` *filename*
: Delete lines in the file that do not contain the any of the substrings in the file.

`DeleteLinesNotMatchingFileItems` *filename*
: Delete lines in the file that do not match the any of the regular expressions in the file.

`DeleteLinesNotStartingFileItems` *filename*
: Delete lines in the file that do not start with any of the substrings in the file.

`DeleteNLines` *quoted-string*
: Deletes up to N lines from the current file, starting from the location of the current line pointer. If the end of the file is reached and less than N lines are deleted, a warning is issued, but editing continues.

`DeleteToLineMatching` *quoted-regex*
: Delete lines from the current position, up to but not including a line matching the regular expression in the quoted string. If no line matches the given expression, a warning is only printed in verbose mode, but all edits are immediately abandoned.

`EditMode "Binary"`
: If set to binary, the file will be edited as if it were a non-ASCII file. See discussion below.

```
EditMode "Binary"
```

`EditSplit` *char*
: Editfiles treats lists as private interpretations, since iteration over lists could lead to unpredictable behaviour if text variables accidentally contain the list separator. The control option `Split` therefore has no effect on editfiles. This definition makes list expansion work in editfiles.

```
EditSplit ","
```

`EmptyEntireFilePlease`
: Deletes all lines from the current file.

`ElseDefineClasses`
: See `DefineClasses`

`EndGroup` Terminates a begin-end conditional structure.

`EndLoop` Terminates a loop. See `ForEachLineIn`

`ExpandVariables`
: This causes cfengine to run through the contents of the file and expand any recognisable editfile strings in the file, if they are defined within the scope of the cfagent script. This gives cfengine an m4 like capacity.

`ExpireAfter` *mins*
`Filter` *filteralias*
: Name a fiter for pruning file searches.

`FixEndOfLine`
: The quoted string which follows may be either '`dos`' or '`Unix`' to fix the end of line character conventions to match these systems. This command should be executed last of all, since cfengine appends new lines with the conventions of the system on which is was complied during edit operations.

`ForEachLineIn` *quoted-filename*
: This marks the beginning of a for-loop which reads successive lines from a named file. The result is like using `SetLine` for each line in the file. Nested loops are not permitted.

`GotoLastLine`
: Moves the file pointer to the last line in the current file.

`HashCommentLinesContaining` *quoted-string*
: Add a '`#`' to the start of any line containing the quoted string.

`HashCommentLinesMatching` *quoted-regex*
: Add a '`#`' to the start of any line exactly matching the quoted regular expression.

`HashCommentLinesStarting` *quoted-string*
: Add a '`#`' to the start of any line starting with the quoted string.

`IfElapsed` *mins*
: As usual.

`IncrementPointer` *quoted-number*
: Increments the value (in lines) of the file pointer by the number of lines specified in the quoted string (as a denary number). e.g. '`''4''`'. Negative values are equivalent to decrementing the pointer. If a request is made to increment/decrement outside of the file boundaries the pointer 'bumps' into the boundary and remains there, i.e. either at start of file or end of file.

`Inform` *quoted-string*
: Set to true or false, on or off to set the inform level for this file. Default is off. Note that, owing to the way that cfengine re-uses code, this feature does not work very well. Detailed messages are not available, only a summary of whether or not the file changed.

`InsertFile` *quoted-string*
: Inserts the named file after the current line position in the file. This should be used in conjunction with a begin-end construction in order to avoid including the file every time cfengine is run. If the file does not exist, or cannot be opened, there is only a warning issued in verbose mode. Note if the file is empty, or if the current line pointer is not set, the file is inserted at the start of the file.

`InsertLine` *quoted-string*
: Inserts the quoted string as a line after the current line pointer in the file. After the insert, the line pointer is incremented by one so that subsequent inserted lines are placed after the first. This should probably be used in conjunction with the conditional begin-end tests to avoid lines being inserted on every run.

`LocateLineMatching` *quoted-regex*
: Moves the current line pointer to the line matching the quoted regular expression. If there is no match, a warning is only issued in verbose mode, but all editing is immediately aborted. See also `WarnIfNoLineMatching` so that you can get an explicit warning, even out of verbose mode.

`PercentCommentLinesContaining` *quoted-string*
: Add a '%' to the start of any line containing the quoted string.

`PercentCommentLinesMatching` *quoted-regex*
: Add a '%' to the start of any line exactly matching the quoted regular.

`PercentCommentLinesStarting` *quoted-string*
: Add a '%' to the start of any line starting with the quoted string.

`Prepend` *quoted-string*
: Add a line containing the quoted string to the start of the file. This should be used in conjunction with the decision structures `BeginGroupIfNoLineMatching` and `BreakIfLineMatches`.

`PrependIfNoLineMatching` *quoted-regex*
: A new version of the older `PrependIfNoSuchLine` with uses a regular expression instead of a literal string. The string prepended is the one set using `SetLine`.

`PrependIfNoSuchLine` *quoted-string*
: Add a line containing the quoted string to the start of the file if the file doesn't contain the exact line already.

`Recurse` *digit/inf*
: For recursive descents when editing whole file trees.

`ReplaceLineWith` *quoted-string*
: Replace the line at the current position with the text in the quoted string. The file pointer remains pointing to this line after the change.

`ReplaceAll` *quoted-regex* `With` *quoted-string*
: Replace all instances of strings matching the regular expression in the first quotes with the exact string in the second set of quotes, throughout the current file. Note that cfengine matches on a left to right basis, with the first match taking precedence, so if your regular expression matches text ambiguously it is the first occurrence which is replaced. For example, if you replace '`cf.*`' with '`CFENGINE`' and cfengine encounters a line '`hello cfengine cfengine`', then this will be replaced with '`hello CFENGINE`' even though two possible strings match the regular expression. On the other hand if the expression is not ambiguous, say replacing '`cfengine`' with '`CFENGINE`', then the result would be '`hello CFENGINE CFENGINE`'.

`ReplaceFirst` *quoted-regex* `With` *quoted-string*
: For each line of the current file, replace the first string matching the regular expression in the first quotes (*quoted-regex*) with the string given in the second set of quotes (*quoted-string*). Matching is done left to right. For example, if you replace '`‘‘YY = [[:digit:]][[:digit:]]’’`' with '`‘‘YY = 04’’`' and cfengine

encounters ‘‘‘YY = 03 but old YY = 70’’’ then it will be replaced with ‘‘‘YY = 04 but old YY = 70’’’

`ReplaceLinesMatchingField` *quoted-number*
: This command replaces any lines in the current file with the current line set by `SetLine` or `ForEachLineIn`, if the lines are split into fields (e.g. the password file) separated by the `SplitOn` character (':' by default), and the corresponding fields match.

  The idea behind this command was to be able to override global passwords (from a file which gets distributed) by new passwords in a local file. Rather than maintaining the files separately, this simply overrides the entries with the new ones.

`Repository` *quoted string*
: This allows a local override of the `Repository` variable, on an item by item basis. If set to “off” or “none” it cancels the value of a global repository.

`ResetSearch` *quoted-string*
: Sets the current-position pointer to the line number in the quoted string. ‘`EOF`’ indicates the end of the file.

`RunScript` *quoted-string*
: Executes the named script command. Before executing the script any edits are saved to disk. After the script has executed, cfengine reloads the file for any further editing operations. The script (which may be any executable program) is appended with two arguments: the name of the file which is being edited and the system hard class (e.g. sun4, ultrix etc.) of the system executing the script.

  CAUTION: cfengine knows nothing about the success or failure of anything that is done during the execution of user scripts. This feature is to be used at the users own peril!

`RunScriptIfLineMatching` *quoted-string*
: Executes the script named with the `SetScript` command only if the current file contains a line matching the quoted regular expression.

  CAUTION: cfengine knows nothing about the success or failure of anything that is done during the execution of user scripts. This feature is to be used at the users own peril!

`RunScriptIfNoLineMatching` *quoted-regex*
: Executes the script named with the `SetScript` command if the current file contains no line matching the quoted regular expression.

  CAUTION: cfengine knows nothing about the success or failure of anything that is done during the execution of user scripts. This feature is to be used at the users own peril!

`SetCommentStart` *quoted-string*
: Specify which string should be used for starting a comment using the commands `CommentLineMatching` and `CommentLineStarting`. The default is the hash symbol ‘`#`’ followed by a single space.

`SetCommentEnd` *quoted-string*
: Specify which string should be used for ending a comment using the commands `CommentLineMatching` and `CommentLineStarting`. The default is the empty string. For example, you could make C style comments by setting CommentStart to '`/*`' and comment end to '`*/`'.

`SetLine` *quoted-string*
: Sets a current line value which can be appended using `AppendIfNoLineMatching` using a regular expression.

`SetScript` *quoted-string*
: Sets the name of a user-supplied script for editing the current file.

`SlashCommentLinesContaining` *quoted-string*
: Add a '`//`' to the start of any line containing the quoted string.

`SlashCommentLinesMatching` *quoted-regex*
: Add a '`//`' to the start of any line exactly matching the quoted regular expression.

`SlashCommentLinesStarting` *quoted-string*
: Add a '`//`' to the start of any line starting with the quoted string.

`SplitOn` *quoted-string*
: This defines a single character which is to be interpreted as a field separator for editing files with columns. The default value for this is '`:`', as is used in the password and group files. It is used in conjunction with `ReplaceLinesMatchingField`.

`Syslog` *quoted-string*
: Set to true or false, on or off to set inform level for this file. Default is off.

`Umask` *quote mode*
: Set local umask for file creation and script execution.

`UnCommentLinesContaining` *quoted-string*
: Uncomment all lines in file containing the quoted string as a substring. The comment delimiters are assumed to be those set using SetCommentStart and SetCommentEnd.

`UnCommentLinesMatching` *quoted-regex*
: Uncomment all lines in file matching the quoted regular expression. The comment delimiters are assumed to be those set using SetCommentStart and SetCommentEnd.

`UnCommentNLines` *quoted-string*
: Uncomments N lines starting from the current position, using the currently defined method for commenting. Note that the comment start and end symbols are removed independently, i.e. they are not matched, so that a comment may be spread over several lines. e.g. If using C style '`/*`' and '`*/`' comments, the command `UnCommentNLines "3"` would uncomment

```
/* 1 */
/* 2 */
```

```
/* 3 */
```

and also

```
/* 1
   2
   3 */
```

`UnsetAbort` *quoted-string*
: Switches off the feature `AbortAtLineMatching`.

`UseShell "false"`
: Normally cfengine uses a shell based exec function to run scripts during editing. This involves the inheritance of environment variables and path, which carries with it an inherent security risk. Setting this value to false causes execution to execute without an encapsulating shell.

`WarnIfFileMissing` *quoted-string*
: If the file to be edited does not exist, a visible alert is issued.

`WarnIfLineContaining` *quoted-string*
: Issue a warning if the quoted string is found as a substring of one or more lines in the file.

`WarnIfLineMatching` *quoted-regex*
: Issue a warning if the quoted regular expression matches one or more lines in the file.

`WarnIfLineStarting` *quoted-string*
: Issue a warning if the quoted string matches the start of one or more lines in the file.

`WarnIfNoLineContaining` *quoted-string*
: Issue a warning if the quoted string is not contained in one or more lines in the file.

`WarnIfNoLineMatching` *reg-ex*
: Issue a warning if the quoted regular expression does not match one or more lines in the file.

`WarnIfNoLineStarting` *quoted-string*
: Issue a warning if the quoted string is not found at the start of one or more lines in the file.

`WarnIfNoSuchLine` *quoted-regex*
: Issue a warning if the quoted regular expression does not match one or more lines in the file.

A limited number of operations can also be performed on purely binary files, e.g. compiled programs, in order to search for strings or viral code, or to modify strings within a program. Binary mode is a mutually exclusive, separate mode to normal editing. The limit on the size of binary files is set by `editbinaryfilesize` in `control`.

`ReplaceAll` *regex* `With` *literal*
: Replaces occurrences of the matched regular expression with\ the provided literal text, only if the length of the literal substitute is less than or equal

to the length of the located string. If the replacement string is shorter, it is padded with ascii spaces (character 32) by default. The padding character can be changed by setting `BinaryPaddingChar` in `control`. Padding with a null byte would lead to corruption of text within a program.

`WarnIfContainsString` *`regex/literal`*
: Yields a warning if the literal string or regular expression matches. Cfengine first attempts a literal match and then a regular expression match.

`WarnIfContainsFile` *`filename`*
: Yields a warning if the contents of the named file exactly match part of the file which is being edited. This can be used to search for binary data which cannot be typed directly into the cfengine program, e.g. virus signatures.

It is suggested that you use these editing functions with caution. Although all possible safeguards have been incorporated into them, it is still possible through carelessness to do damage to important files on your system. Always test editing programs carefully before committing them to your global site configuration.

4.17 files

The `files` facility allows you to touch (create), check for the existence, owner and permissions of files, change the permissions and test for setuid root programs.

4.17.1 Syntax

A files-statement can have several options. We can begin by examining the form of the statement in pseudo-code:

```
files:

   classes::

      /file-object
                        mode=mode
                        owner=uid-list
                        group=gid-list
                        action=fixall/other-options/warnall
                        links=false/stop/traverse/follow/tidy

                        ignore=pattern
                        include=pattern
                        exclude=pattern
                        filter=filter alias
                        xdev=true/on/false/off

                        define=classlist
                        elsedefine=classlist

                        checksum=md5
                        syslog=true/on/false/off
                        inform=true/on/false/off
                        ifelapsed=mins
                        expireafter=mins

   Special OS flags:
                        flags=BSD flags
```

An example would be the following:

```
any::

   /var/spool/printQ  mode=0775  r=0 o=daemon g=daemon  act=fixdirs
```

The meaning of these item is sketched out below and becomes clearer on looking at a number of examples. Note that, each of the options below can be written in either upper or lower case and abbreviated by any unique abbreviation.

/file-object

This is the only obligatory part of a file action. This may be a single file or a directory. If it is a directory then it indicates where does the file search should begin. The recursion specifier may be used to force cfengine to descend into subdirectories in a controlled fashion, starting from this point, checking files there also. The wildcard `home` may also be used. See Section 4.17.4 [home directive], page 89.

A file object is interpreted as a directory if you write it in the following form: '`/directory-path/.`'. i.e. a trailing dot signifies a directory. This then becomes the same as the `directory` command.

`mode=`*`modestring`*

Specifies what the allowed permissions for files are. If cfengine finds that a file's mode is incorrect, the value of the `action` option determines what will be done about it. The modestring should consist of either a three digit octal numbers with '`+`', '`-`' or '`=`' symbols, or a text string like that used by the command `chmod`. For instance: `mode=u=rwx,og+rx` would mean set the read/write and execute flags for the user (file owner) and add the read/execute flags for others and group bits. An example of the numerical form might be `-002` which would mean that the read-for-others flag should either not be set or should be unset, depending on the action you choose. `+2000` would mean that the setgid flag should be present or set, depending on the action. `+2000,-002` would be a combination of these. The '`=`' sign sets to an absolute value, so `=755` would set the file mode to mode 755.

`flags=`*`BSD flags`*

The free BSD Unices have additional filesystem flags which can be seton files. Refer to the BSD `chflags` documentation for this. For example,

```
/tmp/flags.01  mode=0600 owner=0 group=0
               flags=uappnd,uchg,uunlnk,nodump,opaque,sappnd,schg,sunlnk
               action=touch
```

`recurse=`*`number/inf`*

This specifier tells cfengine whether or not to recurse into subdirectories. If the value is zero, only the named file or directory is affected. If the value is 1, it will open at most one level of subdirectory and affect the files within this scope. If the value is `inf` then cfengine opens all subdirectories and files beginning from the specified filename.See Section 4.17.2 [Recursion], page 89.

`owner=`*`owner list`*

This is a list of allowed owners, or uids by number, separated by commas. For example `root,2,3,sysadm`. In cases where you ask cfengine to fix the ownership automatically, the owner will be set to the first recogized owner in the list if and only if it is not one of the named uids in the list.

`group=`*`group list`*

This is a list of allowed groups, or gids by number, separated by commas. For example `wheel,2,3,sysadm`. In cases where you ask cfengine to fix the ownership automatically, the group will be set to the first recognized group in the list if and only if it is not one of the named gids in the list.

`action=`*`action`*

The action is one of the following keywords.

```
warnall warndirs warnplain
 fixall fixdirs fixplain
touch linkchildren create compress alert
```

The upper line results only in warnings being issued. The actions beginning 'fix' prompt cfengine to fix encountered problems without bothering the user. No message is issued unless in verbose mode. The special features on the third line will be explained separately. Alert is like `-print` in the find command, it triggers on the existence of files which have not been ignored, excluded or filtered. This should normally be used together `filter`, in order to locate files of particular types.

`include=`*`wildcard/pattern`*
: You can include this option several times to specify specific patterns which are to be included in the search. Once you specify one pattern you exclude all files not matching at least one of the patterns. The case be useful for restricting a search, or for modifying the permissions of only certain files.

`exclude=`*`wildcard/pattern`*
: You can include this option several times to specify specific patterns which are to be excluded from the search. This overrides any patterns given in the `include=` list.

`ignore`
: This works like the global ignore directive but here you may provide a private list of ignorable directories and files. Unlike include, exclude this affects the way cfengine parses directory trees.

`links=`*`stop/traverse/tidy`*
: Normally cfengine does not descend into subdirectories which are pointed to by symbolic links. If you wish to force it to do so (without using the `-l` command line option) you may give this option the value `true`, or `traverse`, or `follow`. To specify no recursion you set the value `false` or `stop`. Note that the value set here in the cfengine program *always overrides* the value set by the `-l` command line option, so you can protect certain actions from this command line option by specifying a negative value here. If you specify no value here, the behaviour is determined by what you specify on the command line.

The value `links=tidy` has the same effect as the '`-L`' command line option except that here it may be specified per item rather than globally. Setting this value causes links which point to non-existent files to be deleted.

If the warn directive is used (for directories, plain files or both) then only a warning message is issued if the file being tested does not match the specification given. If the fix directives are used then cfengine does not issue a warning, it simply fixes the value silently. Non-existent files are created by the `touch` command. A directory may be touched (created) by writing the filename `/a/b/c/.` with a dot as the last character. (This may also be achieved with the `directories` directive, See Section 4.14 [directories], page 68.)

`define=`*`classlist`*
: If a file operation results in action being taken to fix a file, the colon, comma or dot separated list of classes becomes defined. Warnings do not activate the classes.

`checksum=`*`md5/sha/sha1`*
: If set this option causes cfengine to add a checksum for the named file to a database. Changes in the value of this checksum are then warned as a security issue. This should normally only be used to monitor binary files which one would not expect to change often. Note also that the use of this option can mean a significant performance penalty. Note that it is also possible to use the same database file for cfservd's remote copying by checksum. If you use the same file for both purposes you risk losing warnings. Security warning messages are issued only once and the value in the database is then changed to the new value of the file automatically i.e. the behaviour is similar to that of setuid root program detection, See Section 4.9.10 [ChecksumDatabase], page 39.

`xdev`
: Prevents cfengine from descending into file systems that are not on the same device as the root of the rescurion path.

The default values are `mode=+000`, `recurse=0`, `action=warnall` and any owner or group is acceptable. The default for `links` is to not traverse links unless the `-l` option is set on the command line.

4.17.2 Recursion

The recursion specifier tells cfengine what to do, starting from `/directory name`. A value of `r=0` means 'no recursion' and any checking is limited only to the named file or directory. A value of `r=inf` implies unlimited recursion. Cfengine then descends into all subdirectories checking or setting the permissions of files until it 'bottoms out' at a plain file. A value such as `R=4` means descend recursively into subdirectories, but no more than four levels. This is a useful safety net in preventing unforeseen accidents. A recursive search also bottoms out on device boundaries and symbolic links (provided the *`-l`* option is not used).

4.17.3 Directory permissions

When you specify the permissions for a whole file tree, using the recursion specifier it is awkward to have to remember that directories must be executable. cfengine will do this for you automatically. If you specify that a file tree is to have a read flag set, cfengine will ensure that the corresponding execute flag is also set for directories which live in the tree. So the command

```
files:

  myclass::

      /dir  mode=a+rw r=inf fixall
```

would set all plain files to mode 644 and all directories to 755, that is read/write for everyone on plain files and read/write/execute for everyone on directories.

4.17.4 `home` directive

If you want to check the files of all the users who have their login areas on the current host, you can use a wildcard directive `home` instead of a directory name. In this case the file action iterates over all home directories physically on the current host. The home directories are, of course, located by searching for files which match

```
$(mountpattern)/$(homepattern)
```

i.e. the values which are specified in the `control` part of the program. For example the following line is a very useful service to ignorant users.

```
files:

  any::

    home mode=o-w r=inf act=fixall
```

It ensures automatically that no user has files which can be written to by other arbitrary users.

As a corollary to this, you may write something like

```
  any::

     home/www mode=a+r fixall
```

to specify a special subdirectory of every users' home directory. This statement would check that all of the files in users' world wide web directories were readable for everyone.

4.17.5 Owner and group wildcards

If you do not want to explicitly state the owner or group of a file you may simply omit the group or owner options.

```
  /file-object m=0664 r=inf
```

This example generate a warning if any files under the named directory do not have permission read/write for all users.

4.17.6 Files linkchildren

The `linkchildren` facility is almost identical to that already described under `links`. See Section 4.24.3 [Link Children], page 106. The only difference here is that the ownership and permissions on the links are set all in one operation. For example:

```
myclass::

   /local/lib/emacs m=0770 o=me g=mygroup act=linkchildren
```

4.17.7 touch

The `touch` facility creates a new file with the specified permissions and ownership, or corrects the permissions and ownership of an existing file, in addition to updating the time stamps.

```
myclass::

   /newfile mode=0644 action=touch
```

4.17.8 create

This is like `touch` except that an existing file's time stamps, permissions and ownership will not be modified if the file already exists. If the file does not exist, the attributes are set to the values specified, or to the default values of `0644`.

4.18 filters

A filter is a way of selecting or pruning during a search over files or processes. Since filter rules could apply to several objects, cfengine allows you to define filter conditions as separate objects to be applied in different contexts.

Filter objects can be used in `copy`, `editfiles`, `files`, `tidy` and `processes`. In most cases one writes

```
.. filter=<i>filteralias</i>
```

in the appropriate command. The exception is `editfiles`, where the syntax is

```
{
..
Filter "filteralias"
..
}
```

Example:

```

files:

 /tmp filter=testfilteralias action=alert r=inf

```

Filters are defined in a separate section. Filters for files and processes are defined together. They differ only in the criteria they contain. Here is are examples of file filters:

```
Filters:

  { filteralias1

  Owner:     "mark|cell|motd"
  Group:     "ecg|mark"
  Mode:      "700"

  FromCtime: "date(2000,1,1,0,0,0)"    # absolute date
  ToCtime:   "now"

  FromMtime: "tminus(1,0,0,2,30,0)"    # relative "ago" from now
  ToMtime:   "inf"                     # end of time

  FromAtime: "date(1997,2,22,0,0,0)"
  ToAtime:   "inf"

  FromSize:  "10000"                   # File size interval
  ToSize:    "10mb"

  ExecRegex: "/usr/bin/file $(this) (.*ascii.*)"# Result from "files" command

  Type:      "dir|link"                # reg|link|dir|socket|fifo|door|char|block

  NameRegex: ".*.asc"                  # regex matching file name

  IsSymLinkTo: "/dev/null"             # True if file is a link to object name regex

  Result:    "Type"                    # Result which shouldbe returned

  }
```

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

 { testfilteralias2

 ExecProgram: "/bin/ls $(this)"       # True if the program returns true. $(this) is the cur-
rent object
 }

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

 { testfilteralias3

 Owner: "mark"
 }
```

Filters are evaluated like classes. In fact, the filtering works by evaluating the class attributes for each file.

File filters:

`Owner:` and Group can use numerical id's or names, or `''none''` for users or groups which are undefined in the system passwd/group file.

Mode: applies only to file objects. It shares syntax with the `mode=` strings in the files command. This test returns true if the bits which are specified as 'should be set' are indeed set, and those which are specified as 'should not be set' are not set.

`Atime:,Ctime:,Mtime:`
apply only to file objects. These specify ranges From and To. If the file's time stamps lie in the specified range, this returns true. Times are specfied by a six component vector

```
(year,month,day,hour,minutes,seconds)
```

This may be evaluated as two functions: date() or tminus() which give absolute times and times relative to the current time respectively. In addition, the words now and inf may be used. e.g.

```
FromCtime: "date(2000,1,1,0,0,0)"   # absolute date
ToCtime:   "now"

FromMtime: "tminus(1,0,0,2,30,0)"     # relative "ago" from now
ToMtime:   "inf"                     # end of time
```

`Type:` applies only to file objects may be a list of file types which are to be matched. The list should be separated by the OR symbol '`|`', since these types are mutually exclusive. The possible values are currently

```
file|reg|link|dir|socket|fifo|door|char|block
```

Note that `file` and `reg` are synonymous.

`ExecRegex:`
matches the test string against the output of the specified command.

`NameRegex:`
matches the name of the file with a regular expression.

`IsSymLinkTo:`
: applies only when the file object $(this) is a symbolic link. It is true if the regular expression matches the contents of the link.

`ExecProgram:`
: matches if the command returns successfully (with return code 0). Note that this feature introduces an implicit dependency on the command being called. This might be exploitable as a security weakness by advanced intruders.

`Result:` specifies the way in which the above elements are combined into a single filter.

Process filters:

`Owner` process owner UID (quoted regex)

`PID:` process ID (quoted regex)

`PPID:` parent process ID (quoted regex)

`PGID:` process group ID (quoted regex)

`RSize:` resident size (quoted regex)

`VSize:` virtual memory size (quoted regex)

`Status:` status (quoted regex)

`Command:` CMD or COMMAND fields (quoted regex)

`(From/To)TTime:`
: Total elasped time in TIME field (accumulated time)

`(From/To)STime:`
: Starting time for process in STIME or START field (accumulated time)

`TTY:` terminal type, or none (quoted regex)

`Priority:`
: PRI or NI field (quoted regex)

`Threads:` NLWP field for SVR4 (quoted regex)

`Result:` logical combination of above returned by filter (quoted regex)

Examples: processes started between 18th Nov 2000 and now.

```
{ filteralias

FromSTime: "date(2000,11,18,0,0,0)"
ToSTime:   "now"
}
```

All processes which have accumulated between 1 and 20 hours of CPU time.

```
{ filteralias

FromTTime:  "accumulated(0,0,0,1,0,0)"
ToTTime:    "accumulated(0,0,0,20,0,0)"
}
```

4.18.1 Complete filter examples

Here is an example filter to search for all files which are either directories or links, or any kind of file owned by mark, in group cfengine.

```
control:

 actionsequence = ( files )

files:

 /tmp       filter=testfilteralias action=alert r=inf
 /cfengine  filter=testfilteralias action=fixall r=inf mode=644

filters:

  { testfilteralias

  Owner:     "mark"
  Group:     "cfengine"
  Type:      "dir|link"

  Result:    "Type|(Owner.Group)"  # Both owner AND group required correct
  }
```

Find all ELF executables using data from the Unix `file` command. Caution, this takes a long time if used indescriminately.

```
control:

 actionsequence = ( files )

files:

 /tmp       filter=testfilteralias action=alert r=inf
 /cfengine  filter=testfilteralias action=fixall r=inf mode=644

filters:

  { testfilteralias

  ExecRegex: "/bin/file (.*ELF.*)"

  Result: "ExecRegex"
  }
```

Here is an example which warns of any process coupled to a terminal started in November:

```
control:

 actionsequence = ( processes )

filters:

  { filteralias
  FromSTime: "date(2000,11,0,0,0,0)"
  ToSTime:   "date(2000,11,30,0,0,0)"
  TTY: ".*pt.*"
  Result: "TTY.STime"
```

```
  }

processes:

 "." filter=filteralias action=warn
```

4.19 groups/classes

The `groups` action (equivalently referred to as `classes` as of version 1.4.0) is used to define classes which stand for groups of hosts. If you use the NIS (network information service) facility for defining *netgroups* then this idea will already be familiar to you and you can probably use your already-defined netgroups in cfengine.

To define a group, you simply make a list and assign it a name. Here is an example of the syntax:

```
groups:

  ANDed_class::

    science = ( saga tor odin )

    packages = ( saga )

    AllHomeServers   = ( saga )
    AllBinaryServers = ( saga )

    OIH_servers = ( saga )
    OIH_clients = ( tor odin )

    notthis = ( !this )

    ip_in_range = ( IPRange(129.0.0.1-15) )  # host is in ip address range
    ip_in_range = ( IPRange(129.0.0.1/24) )  # host is in ip address range (CIDR notation)

    compute_nodes = ( HostRange(cpu-,1-32) ) # host name in the cpu-01 through cpu-32 range
```

To include a list of hosts from a NIS netgroup, you use the '`+`' symbol, or the '`+@`' construction. For example:

```
groups:

   science = ( +science-allhosts )

   physics = ( +physics-allhosts )

   physics_theory = ( +@physics-theory-sun4 dirac feynman schwinger )
```

Using an enormous netgroup does not use up any space. A group declaration results in the storage of only the class name regardless of how many hosts are in the list. The rule is that the left hand side of the assignment becomes defined (true) if the list on the right hand side includes the host which is parsing the file — i.e. `$(host)`.

In some cases your netgroups will not correspond exactly to the list you want, but it might be more convenient to use a netgroup *except* for certain hosts. You can 'undefine' or remove hosts from the netgroup list by using the minus '-' symbol. For example:

```
group = ( +mynetgroup -specialhost -otherhost )
```

which means, of course, all hosts in netgroup `mynetgroup` except for `specialhost` and `otherhost`. Finally, you may also subtract two netgroups in the following manner.

```
group = ( +bignetgroup -smallnetgroup )
```

The 'minus' command effectively eliminates its members from `bignetgroup` if they exist within that group. If none of the hosts in `smallnetgroup` exist in `bignetgroup` then the command has no effect.

Groups may contain previously defined cfengine groups too. This allows one class to inherit the attributes of another class, for instance:

```
AllSun4Hosts   = ( sonny sunny solar stella )
AllUltrixHosts = ( ully olly wally golly )

AllBSD = ( AllSun4Hosts AllUltrixHosts )
```

The classes on the right hand side are effectively ORed together into the left hand side. This enables complex classes to be constructed from several other basic classes, e.g.

```
SpecialTimes = ( Hr00 Monday Day1 )
```

which evaluates to true every day when it between 00:00 hours and 00:59, all day Monday and all day on the first day of every month.

If you apply a class predicate before a definition then the result is effectively the AND of the classes:

```
Hr00::

 SpecialTime = ( Monday Tuesday )
```

defines `SpecialTime` at Hr00 on Monday or Tuesday.

Finally, you can define groups (strictly classes) by the result of a shell command. A shell command or program is deemed to be 'true' if it exits with a status of zero, i.e. it calls `exit(0)`. Any other value is taken to be false. You can include shell commands as the members of groups in order to define classes based on the outcomes of your own scripts by enclosing the script in single or double quotes:

```
have_cc = ( '/bin/test -f /usr/ucb/cc' )
```

The class `have_cc` will then be defined if the shell command returns true. Of course, you can put any script or program in the single quotes as long as they adhere to the convention that zero exit status means true. If you have several members which are shell commands, then the effect is to make the class the logical OR of the scripts' results.

4.20 homeservers

The `homeservers` declaration need only be used if you are using cfengine's model for mounting NFS filesystems. This declaration informs hosts of which other hosts on the network possess filesystems containing home directories (login areas) which client hosts should mount.

A sample homeserver declaration looks like this:

```
homeservers:

   Physics::  einstein
   Math::     riemann euler
```

The meaning of this declaration is the following. Any host which finds itself to be a member of the classes on the left hand side of the assignment need to mount all home directory resources from the hosts on the right hand side of the assignment. The pattern variable `homepattern` is used to determine which resources are home directories in the list of `mountables`. See Section 4.28 [mountables], page 117.

Let us consider an example in which `homepattern` is set to the wildcard value '`home?`' and the mountables list is given by

```
mountables:

   einstein:/mysite/einstein/home1
   einstein:/mysite/einstein/home2  mountoptions=soft,bg,intr,rsize=8192,wsize=8192
   riemann:/mysite/riemann/local    readonly=true
   euler:/mysite/euler/home1
```

Any host in the group `Physics` would now want to mount all home directories from the host `einstein`. There are two of these. Both the filesystems listed for `einstein` match the `homepattern` variable since they end in '`home?`'. cfengine would therefore take this to mean that all hosts in `Physics` should mount both of these filesystems.

Hosts in `Math`, on the other hand, should mount only homedirectories from the hosts `riemann` and `euler`. There is only a single filesystem on `riemann` and it does not match `homepattern`, so it is not mounted. On `euler` there is a match, so this filesystem will be added to the appropriate hosts.

Cfengine picks out home directory resources from the `mountables` *list by trying to match the* `homepattern` *variable, starting from the end of the directory name. You do not therefore have to use the designation* `/site/host/home?` *but this is a simple choice and is highly recommended.*

4.21 ignore

When you specify a recursive search as part of a `files`, `tidy` or `copy` action, you would sometimes like to exclude certain directories from the list of sub directories. In most cases you will want to do this on a per-command basis (see the pages for these actions separately), but you can also make a global ignore list. This can be accomplished by adding the directory to the ignore-list. The syntax is

```
ignore:

   wildcards/directories/filenames
```

For example:

```
ignore:

   any::

      #
      # Prevent tidying .X11 directories in /tmp where
      # window managers write semaphores
      #

      .X11

      #
      # Don't tidy emacs locks
      #

      !*
      /local/lib/gnu/emacs/lock/
      /local/tmp
      /local/bin/top
      /local/lib/tex/fonts
      /local/etc
      /local/www
      /local/mutils/etc/finger.log
```

None of the above directories will be checked or entered during recursive descents unless a specific command is initiated to search those directories with their names as the top of the search tree.

A handy tip if you are tidying '`/tmp`' recursively is to include the directory '`.X11`' here. This directory is used by the X-windows system and deleting it while a window manager has an open session can cause the user some trouble.

Ignore refers to all recursive searches in tidy, files, copy and links.

4.22 import

To break up a large configuration file into smaller files you can use the include directive. This conditionally reads in files if the class on the left hand side of the assignment matches the host parsing the file. This enables also a variety of cfengine configuration scripts to read in a standard set of default settings. The syntax of the statement is:

```
import:

   any::

      cf.global_classes


   linux::

      cf.linux_classes
```

Note that, if you define variables in an imported file they will not be defined for operations in their parent files. This because cfengine reads in all the import files after the main file has been parsed—not at the place where you call import in your script. This means that variables or macros defined in imported files are only defined after the main program. Variables from earlier files are inherited by later includes, but not *vice-versa*.

4.23 interfaces

```
interfaces:

  classes::

    interfacename netmask=netmask broadcast=broadcast
```

If you have more than one network interface, or you do not wish to use the default interface name, this section may be used to define further interfaces to be checked. This feature can replace the older method of setting netmask and broadcast address in `control:`. If the `netmask` variable is not set, cfengine ignores the default interface configuration. Example:

```
interfaces:

  "le1" netmask=255.255.255.0 broadcast=ones
  "le2" netmask=255.255.255.0 broadcast=ones
```

4.24 links

The symbolic links function is one of the greatest plusses in cfengine as a system administration tool. It allows you to do two things: check single links for correctness and consistency (or make them if they do not exist), and check or make links to every file in a designated directory. This latter feature is called multiple linking or linking children. The `linkchildren` feature is also available from the `files` action See Section 4.17 [files], page 86. The syntax of a link item is:

```
 from-link ->[!] to-object
or
 from-link +>[!] to-object

          type=symbolic/absolute/abs/hard/relative/rel
          copy=pattern
          recurse=number/inf/0
          copytype=checksum/ctime
          include=pattern
          exclude=pattern
          ignore=pattern
          action=silent
          deadlinks=kill/force
          define=classlist
          nofile=kill/force
          syslog=true/on/false/off
          inform=true/on/false/off
          ifelapsed=mins
          expireafter=mins
```

The special variable `$(binserver)` *can be used in* `links`.

4.24.1 Single links

To define a single link, you create an entry of the following form:

```
links:

  class::

     linkname -> object_to_link_to
     linkname -> ./relative_link
     linkname -> ../relative_link
```

If links exists and point to their intended destinations then no action is taken. If a link exists but points incorrectly then a warning is issued, unless the pling operator '`!`' is given, in which case the correct value is forced. If the link exists and points to a file which does not exist a warning is issued unless the command line option `-L` is used, in which case the link is deleted.

Here is an example of some valid link statements.

```
links:
```

```
Physics.sun4::

 /usr/local       -> /$(site)/$(host)/local
 /home            -> /$(site)/$(host)/u1
 /etc/sendmail.cf -> /usr/local/mail/etc/global-sendmail.cf

 /usr/lib/sendmail ->! /local/lib/sendmail
```

cfengine makes any directories which are required leading up to the link name on the left hand side of the arrow automatically. In the last example the 'pling' forces cfengine to make the link even if a file for link exists previously. Plain files are saved by appending '`.cfsaved`' to the filename, or by moving to a repository, whereas old links are removed. The same effect can be enforced globally using the `-E` option, but only if the program is run interactively. (In this case a prompt is issued to make sure that you wish to use such a big hammer on your system!)

The link operation accepts a number of parameters

`type=`*`hard/relative/absolute`*
: If the link type is hard, a hard link is created See Section 4.24.5 [Hard links], page 107. Symbolic links may specify two special types. If `relative` is selected, and the 'to' object is an absolute path name, the link name will be rewritten as a pathname relative to the source file, using '`.`' and '`..`' to move relative to the current directory. For instance, a link from '`/usr/local/file`' to '`/usr/file`' would be linked as '`./../file`'. If the 'to' object is already relative, this has no effect.

  If `absolute` is specified, cfengine will try to resolve the true path location of the 'to' object, expanding any symbolic links or dots in the path name, up to a maximum of four levels of symbolic links.

`copy=`*`pattern`*
: This option can be repeated any number of times to build up a list of filenames or wildcards which are to be copied rather than linked symbolically. The copy is made on an age-comparison basis. A global variable may also be set to invoke this feature See Section 4.9.15 [copylinks], page 40. Directories cannot be copied in this way.

`copytype=`*`checksum/ctime`*
: This specifies the basis for deciding whether to update a file which is to be copied instead of linked See Section 4.11 [copy], page 58.

`nofile=`**kill**`/`*`force`*
: This decides what happens to links which point to non-existent files. The default action is to remove such links, or refuse to create them. By setting the *force* option you can force cfengine to make symbolic links to files which do not exist. This is useful for setting up links to filesystems which are not permanently mounted.

`exclude=`*`pattern`*
: This option can be repeated any number of times to build up a list of filenames or wildcards which are to be excluded from the linking process. A global variable may also be set to invoke this feature See Section 4.9.30 [excludelinks], page 43.

`ignore` This works like the global ignore directive but here you may provide a private list of ignorable directories and files. Unlike include, exclude this affects the way cfengine parses directory trees.

`recurse=`*`number`*`/inf`
: This option can only be used with multiple link operations See Section 4.24.2 [Multiple Links], page 105. If this option is specified, cfengine links only non-directory objects. Directories are instead created and links within those directories are also created. The value of this option specifies the maximum number of levels to which cfengine should recursively descend a link tree. `inf` means infinite recursion. Cfengine also ignores files and directories in the ignore list See Section 4.21 [ignore], page 100.

`define=`*`classlist`*
: If a link is created or replaced, the colon, comma or dot separated list of classes becomes defined.

The final feature of the links facility is connected to the use of the cfengine model for mounting NFS filesystems. In particular it concerns the variable `$(binserver)`. The easiest way to understand this feature is to illustrate a couple of examples. Consider the following:

```
links:

  any::

    /local -> /${site}/${binserver}/local
```

The result of this command is quite different depending on which host is executing it. The variable `$(site)` clearly has a fixed value, but the variable `$(binserver)` might expand to any valid binary server for the host executing the program. See Section 4.7 [binservers], page 30. The procedure cfengine adopts is to go through its list of mountables, keeping only those mountable resources which belong to defined binary servers for the current host. It then attempts to match a filesystem by substituting `$(binserver)` with each of its valid binservers in turn and it matches the first one binary server which yields an existing file.

Note that every host is a binary server for itself, so that the value of `$(binserver)` which has absolute priority is alway the same as the value of `$(host)`. This ensures that the link will always be made to a local filesystem if the rules of the model are upheld.

4.24.2 Multiple Links

With the link symbol `+>`, you opt to link all of the files in a directory to corresponding files in another directory. This procedure is sometimes useful for installing software. In the example

```
links:

 myclass::

   /usr/local/bin +>  /usr/local/lib/perl/bin
   /opt           +>! /local
```

every file in the directory `/usr/local/lib/perl/bin` is linked symbolically to a corresponding file in `/usr/local/bin`. The 'pling' character forces cfengine to replace old links or plain files already existing. Old links are removed, whereas old files are saved by appending '.cfsaved' to the filename See Section 4.9.51 [repository], page 50.

Each time cfengine runs it goes through all of the files in the directory concerned and checks the appropriate link accordingly. If new files appear, new links will be added. If a file disappears but the link to it remains, a warning will be issued, unless the -L command line option is used, in which case the link is deleted.

4.24.3 Link Children

The linkchildren directive is a closely related to the cfengine model for NFS filesystems. It is a way of making links which embodies a rudimentary kind of 'intelligence'.

Consider the following:

```
links:

   any::

      /usr/local/lib/emacs +> linkchildren
```

The word `linkchildren` automatically tells cfengine that it should look for an appropriate file to link to on a binary server for the current host. The exact meaning of the above statement is as follows. cfengine begins searching though the list of mountable resources, discarding any filesystems which do not belong to valid binary servers. It looks for a filesystem ending in 'emacs' (the last link of the left hand side). If all is well, these file systems are already mounted and they can be searched. If no resource is found ending in 'emacs', we go to the next link `lib` and look for a filesystem ending in 'lib'. If this is not found we go to `local` and so on. When a match is made, cfengine then tries to locate the file by checking whether it exists relative to the matched filesystem. For example, suppose 'local' matched with `host:/site/host/local`. It would then try to locate `host:/site/host/local/lib/emacs` and link all of the children therein to the local file directory `/usr/local/lib/emacs`.

Here is another example which makes reference to the cfengine model for mounting NFS filesystems. Suppose you have a host with some spare disk space. You want to mount `/usr/local` from the binary architecture server, but you also want to use the disk you have locally. The following lines

```
links:

   electron::

      /$(site)/electron/local +> linkchildren

   any::

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

have the effect of creating a directory `/$(site)/electron/local` and filling it with links to all of the files and directories on the binary server's mounted filesystem. It results in an exact copy (by linkage) on the local disk, but does not use up your local disk space. The space you have remaining could, for example, be used for software with a special license for that host. The second link links `/usr/local` to the 'nearest' binary server. But the nearest binary server is always `$(host)` which means this evaluates to a file which now exists because of the first command, so on the host 'electron' the directory `/usr/local` ends up being a link to `/$(site)/electron/local` which is full of links to the binary server.

If you've caught your breath after that mouthful you probably have mixed feelings about creating a bunch of links in this way. What happens if the files they point to are removed? Then you are left with a lot of useless links. Actually this is no problem for cfengine, since you can ask cfengine to simply remove links which point to non-existent files See Section 4.17 [files], page 86. Nevertheless, this feature clearly requires some caution and is mainly a spice for advanced users of the cfengine model.

4.24.4 Relative and absolute links

When specifying symbolic linking, you can ask cfengine to change the link type to be either relative to the source or to be an absolute path. What this means is the following. Consider the following link:

```
/var/tmp/cfengine -> /local/cfengine
```

If we add the option `type=relative`, then instead of creating a link which points to '`/local/cfengine`', the link is created pointing to the location

```
./../../local/cfengine
```

In other words, the link is relative to the calling directory '`/var/tmp`'.

If a link is specified as being absolute with the option `type=absolute`, then cfengine attempts to resolve to value of the link so as to be the true path of the target. If the target name contains a symbolic link, then this is expanded as far as possible to give the true path to the file. For example, if '`/local`' is really a link to '`/site/myhost/local`' then the link would point to '`/site/myhost/local/cfengine`'.

4.24.5 Hard Links

Cfengine will also allow you to create hard links to regular files. A hard link is in every way identical to the original file, it merely has a different name (technically, it is a duplicate inode). To create a hard link you use the link-option `type=hard`. For example:

```
links:

   /directory/newname -> /directory/othername type=hard
```

Cfengine will not create hard links to directories or other special files. This is always a slightly dubious practice and is best avoided anyway. POSIX says that the hard link can be on a different device to the file it points to, but both BSD and System V restrict hard links to be on the same device as their predecessors. Cfengine has no policy on this, but—in the theoretical case in which the hard link and the predecessor were on different file systems—it becomes near impossible to determine with certainly between a hard link and a very similar regular file, and thus cfengine issues a warning in verbose mode about this eventuality. Provided both link and predecessor are on the same filesystem cfengine determines the status of hard links by comparing the device and inode numbers of the file pointed to.

4.25 mailserver

The `mailserver` declaration need only be used if you are using cfengine's model for mounting NFS filesystems. This declaration informs hosts of which NFS filesystem contains mail for its users. All hosts apart from the mail-host itself must then mount the mail spool directory across the network. The declaration looks like this:

```
mailserver:

   class::      mailhost:/var/spool/mail
```

The result of the `mailcheck` command in the action-sequence is now to mount the filesystem `/var/spool/mail` on the host `mailhost`. This action is carried out on any machine which does not already have that filesystem mounted.

The mail spool directory is mounted, by default, onto the official mail spool directory for the system which is parsing the program. In other words, on an HPUX system, the spool directory is mounted on `/usr/mail` by default, whereas on a Sun system it would be mounted on `/var/spool/mail`. The default location can be changed by using the resource file. See Section 7.2 [cfrc resource file], page 154.

4.26 methods

From version 2.1.0, cfagent provides for the execution of closed functions or "methods". Methods are similar to the old idea of modules, but they are implemented in a way that allows collaboration between different hosts within a network, using a common standard. Methods must be cfengine programs however, wheras the module interface can be written in any script language.

Methods are designed to offer a firewall-like proxy interface for remote method execution. We can call methods a form of 'voluntary RPC', in which hosts execute methods for one another on a purely voluntary basis. This builds in anti-spamming protection. The principle used is that hosts should be immune to Denial of Service attacks; they should only be able to disadvanatge themselves with the attempt.

(Remote method execution was not implemented until version 2.1.3. It is still considered experimental and is not recommended for large production environments until this paragraph is removed from the documentation.)

Methods allow you to call an independent cfengine program, pass it arguments and classes, and collect the results for use in your main program. It thus introduces parent-child semantics into cfengine "imports". A method is more than an import. (Import is analagous to a C #include, while a method is like a C function.) Communication is peer to peer, by mutual consent. There is no "method server" that executes methods on remote hosts. Hosts exchange information by invitation only. This is an unreliable service (in the sense of UDP).

The order of method exeuction is not guaranteed. This results from the decoupling between client request and service provision.

```
methods:

   class::


    function_name(parameters or none)

    action='filename'

    sendclasses=comma separated class list

    returnvars=comma separated variable list or void
    returnclasses=comma separated class list

    server=ip-host/localhost/none

    forcereplyto=ip address

    owner=setuid
    group=setgid
    chdir=cd for child
    chroot=sandbox directory
```

Most of these functions will be familiar from other cfengine commands. Some special ones are noted below:

`action` The name of the method file that should be defined in the modules directory of the server host.

`forcereplyto`
: Sometimes nameservice problems (especially with remote devices) can lead to confusion about where a method should be sent. The caller can therefore declare to the server which address it wants the reply to be marked for.

`returnvars`
: Returns the values of the variables to the parent process. This acts as an access control list for variable names transmitted by the child process. The names returned by the child must match this list.

`returnclasses`
: Returns the classes to the parent process, if and only if they are defined at the end of the current method. This acts as an access control list for class names transmitted by the child process. The names returned by the child must match this list.

`sendclasses`
: Transmits the current status of the named classes to the child process. In other words, if the listed classes are defined, they become defined in the child process, else they remain undefined. The class may still be defined in the child by independent local definitions.

If the server is set to `localhost`, the method will be evaluated on the local machine. If the server is set of `none`, the method will not be executed at all.

The function arguments may not be empty, but a null value can be transmitted with a dummy value, e.g. `Function(null)` or `function(void)`. Here is an example method call.

```
 # cfagent.conf

control:

   actionsequence = ( methods )

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

methods:

 any::

   SimpleMethod(null)

      action=cf.simple
      returnvars=null
      returnclasses=null
      server=localhost
```

With method file (located in the ModulesDirectory),

```
 # cf.simple
```

```
  control:

    MethodName       = ( SimpleMethod )
    MethodParameters = ( null )
    actionsequence   = ( timezone )


  classes:

     dummy = ( any )

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

alerts:

 dummy::

  "This simple method does nothing"

  ReturnVariables(void)
  ReturnClasses(void)
```

On executing this example, the output is:

```
nexus$ ./cfagent -f ./cftest
cfengine:myhost:SimpleMethod: cfengine:nexus: This simple method does nothing
```

If the server name is a wildcard, e.g. * then this acts as a multicast or broadcast.

4.26.1 Localhost examples

The following example collects the tar file, unpacks it, configures and compiles it, then tidies its files.

```
#####################################################
#
# This is a cfengine file that calls a method.
# It should be in the usual place for cfinputs
#
#####################################################

control:

   actionsequence = ( methods )

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

methods:

   InstallTar(cfengine-2.1.0b7,/local/gnu)

      action=cf.install
      returnvars=null
      returnclasses=null
      server=localhost
```

We must install the method in the trusted modules directory (normally /var/cfengine/modules or WORKDIR/modules).

```
#####################################################
#
# This is an example method file, that needs to be
# in the module directory /var/cfengine/modules
# since this is the trusted directory
#
# e.g. InstallFromTar(cfengine-2.1.0,/usr/local/gnu)
#
#####################################################

control:


 MethodName       = ( InstallTar )
 MethodParameters = ( filename gnuprefix )

 path = ( /usr/local/gnu/bin )


 TrustedWorkDir = ( /tmp )

 TrustedSources = ( /iu/nexus/ud/mark/tmp )
 TrustedSourceServer = ( localhost )

 actionsequence = ( copy editfiles shellcommands tidy )

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

classes:

  Force = ( any )

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

copy:

 $(TrustedSources)/$(filename).tar.gz

    dest=$(TrustedWorkDir)/$(filename).tar.gz
    server=$(TrustedSourceServer)

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

shellcommands:

 "$(path)/tar zxf $(filename).tar.gz"

     chdir=$(TrustedWorkDir)

 "$(TrustedWorkDir)/$(filename)/configure --prefix=$(gnuprefix)"

    chdir=$(TrustedWorkDir)/$(filename)
    define=okay

 okay::

 "$(path)/make"
```

```
     chdir=$(TrustedWorkDir)/$(filename)

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

tidy:

  $(TrustedWorkDir) pattern=$(filename) r=inf rmdirs=true age=0


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

#editfiles:
#
#{ $(TrustedWorkDir)/$(filename)/configure-opts
#
#AppendIfNoSuchLine "Something ???"
#}

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

alerts:

 Force::

  ReturnVariables(none)
  ReturnClasses(success)
```

A more complex example is given below:

```
GetAnalysis("${parent1}",param2,ReadFile("/etc/passwd",300))

  # The name of the method that is in modulesdir

  action=cf.methodtest

  # The variables that we get back should be called these names
  # with method name prefix

  returnvars=a,b,c,d

  # This is an access list for returned classes. Classes will
  # only be handed back if they are included here

  returnclasses=define1,define2,class1

  # The host(s) that should execute the method

  server=localhost

  # Only localhost can decide these - not a remote caller
  #    owner=25
  #    group=root
  #    chdir=/var/cfengine
  #    chroot=/tmp
```

Here the function being called is the cfengine program '`cf.methodtest`'. It is passed three arguments: the contents of variable *parent1*, the literal string "param2" and the first

300 bytes of the file '`/etc/passwd`'. On return, if the method gets executed, the values will be placed in the four variables:

```
$(GetAnalysis.a) $(GetAnalysis.b) $(GetAnalysis.c) $(GetAnalysis.d)
```

If the classes `define1` etc, are returned by the method, then we set them also in the main program as

```
GetAnalysis_define::
```

In other words, the class name is also prefixed with the method name to distinguish it. (`returnclasses` works like an access control list for setting classes, deciding whether or not the main script should accept the results from the child method.). The remaining options are as those for executing shell commands, and apply only on the host that executes the function.

Both the client and server hosts must have a copy of the same method declaration. The client should have a non-empty `server=` declaration. The server side should have no `server=` declaration unless it is sending the request on recursively to other hosts. At present only requests to localhost are allowed, so only there is automatic access to the rule.

The cfagent file that contains the method code must have the following declarations:

control:

MethodName = (*identifier*) MethodParameters = (*spaced list of recipient variables or files*)

....

alerts:

Return variables are alerts to parent

ReturnVariables(*comma separated list of variables or functions or void*) ReturnClasses(*comma separated list of classes*)

e.g.

```
control:

 MethodName       = ( GetAnalysis )
 MethodParameters = ( value1 value2 /tmp/file1 )

 # ....

alerts:

  # Return variables are alerts to parent

  ReturnVariables("${var1}","${var2}","var3",literal_value)
  ReturnClasses(class1,class2)
```

The parameters transmitted by the parent are read into the formal parameters `value1`, `value2` and the the file excerpt is placed in the temporary file '`/tmp/file1`'.

The return classes are passed in their current state to the parent; i.e. if class1 is defined then it is offered to the parent, but if it is not defined in the method, it is not passed on. The parent can then choose to accept or ignore the value.

4.26.2 Remote host examples

Methods can also be scheduled for execution on remote hosts.

- Both hosts must have an identical copy of the method stanza
- Public keys must be exchanged between the cooperating hosts
- Access must be granted to '`/var/cfengine/rpc_out`' in cfservd.

Remote method execution is the same as local method execution except for some additional requirements. A list of collaborating peers must be added to the control section of '`update.conf`'. In order the the requests to be collected automatically, you must have `copy` in the action sequence of the '`update.conf`' file.

```
control:

 MethodPeers = ( hostname list ) # Must have copy in actionsequence
```

This list tells the agent which remote hosts to collaborate with, i.e. whom should we contact to look for work that we have promised to perform? In order the the requests to be collected automatically, you must have `copy` in the action sequence of the '`update.conf`' file.

For example, to make two hosts collaborate:

```
methods:

 host1|host2::

  MethodTest("my test!")

  action=cf.methodtest
  server=host2.iu.hio.no
  returnclasses=null
  returnvars=retval
  ifelapsed=120
```

Note that an important aspect of remote method invocation is that there is only voluntary cooperation between the parties. A reply bundle from a finished method can collected from a server by the client many times, causing the classes and variables associated with it to be defined at regular intervals, controlled by the ifelapsed time. To avoid multiple actions, you should lock methods or their follow-up actions with long ifelapsed times. This is a fundamental 'feature' of voluntary cooperation: each party must take responsibilty for the sense of what it receives from the other. This feature will not be to everyone's taste, and it is unconventional. However, voluntary cooperation provides a way of collaborating without trust in a framework that forces us to confront the security issues directly. As such, it is a successful experiment.

4.27 miscmounts

If you do not use the cfengine model for statically mounting NFS filesystems (or if there are filesystems which do not naturally fall into the bounds of that model) then you can still statically mount miscellaneous filesystems using a statement of the form:

```
miscmounts:

   class::

      infohost:source-directory destination mode

      infohost:source-directory destination mode=mode

     ifelapsed=mins expireafter=mins
```

For example

```
physics::

 # old syntax

   libraryserver:/$(site)/libraryserver/data1
                     /$(site)/libraryserver/data1 ro

 # consistent syntax

   libraryserver:/$(site)/libraryserver/data2
                     /$(site)/libraryserver/data2 mode=ro

   host:/foo /foo mode=rw,bg,hard,intr
```

This statement would mount the directory '`/$(site)/libraryserver/data`' physically attached to host libraryserver onto a directory of the same name on all hosts in the group `physics`. The modes `ro` and `rw` signify read-only and read-write respectively. If no mode is given, read-write is assumed.

4.28 mountables

The `mountables` declaration need only be used if you are using cfengine's model for mounting NFS filesystems. This declaration informs hosts of what filesystem resources are available for mounting. This list is used in conjunction with `binservers` and `homeservers` to determine which filesystems a given host should mount, according to the cfengine model.

The syntax of the list is:

```
mountables:

   class::
       "filesystem to mount"
   readonly=false/off/true/on
   mountoptions=nfs-options
```

e.g.

```
mountables:

   class::

   server:/site/server/u1
   server:/site/server/local
   linuxhost:/site/linuxhost/local
   linuxhost:/site/linuxhost/u1
```

Notice that binary and home-directory filesystems are mixed freely here. Cfengine determines which of the entries are homedirectories using the `homepattern` variable.

Every time you add a disk or a mountable partition to your network, you should add the partition to the list of mountables.

NOTE: This list is read in order, top down. Cfengine looks for the first filesystem matching a given binary server when expanding the variable `$(binserver)`*, so sometimes the ordering of filesystems matters.*

This list can be accessed in editfiles, to allow straightforward configuration of the automounter, using the command `AutomountDirectResources`.

4.29 processes

Using the processes facility, you can test for the existence of processes, signal (kill) processes and optionally restart them again. Cfengine opens a pipe from the system ps command and searches through the output from this command using regular expressions to match the lines of output from 'ps'. The regular expression does not have to be an exact match, only a substring of the process line. The form of a process command is

```
processes:

  class::

    "quoted regular expression"

                        restart "shell command"
                        useshell=true/false/dumb
                        owner=restart-uid
                        group=restart-gid
                        chroot=directory
                        chdir=directory
                        umask=mask

                        signal=signal name
                        matches=number
                        define=classlist
                        elsedefine=classlist

                        action=signal/do/warn/bymatch
                        include=literal
                        exclude=literal
                        filter=filter_name

                        syslog=true/on/false/off
                        inform=true/on/false/off
                        ifelapsed=mins
                        expireafter=mins

    SetOptionString "quoted option string"
```

By default, the options sent to ps are "-aux" for BSD systems and "-ef" for System V. You can use the `SetOptionString` command to redefine the option string. Cfengine assumes only that the first identifiable number on each line is the process identifier for the processes, so you must not choose options for ps which change this basic requirement (this is not a problem in practice). Cfengine reads the output of the ps-command normally only once, and searches through it in memory. The process table is only re-consulted if `SetOptionString` is called. The options have the following meanings:

`signal=`*`signal name`*

This option defines the name of a signal which is to be sent to all processes matching the quoted regular expression. If this option is omitted, no signal is sent. The signal names have the usual meanings. The full list, with largely standardized meanings, is

```
hup       1   hang-up
```

```
int       2   interrupt
quit      3   quit
ill       4   illegal instruction
trap      5   trace trap
iot       6   iot instruction
emt       7   emt instruction
fpe       8   floating point exception
kill      9   kill signal
bus      10   bus error
segv     11   segmentation fault
sys      12   bad argument to system call
pipe     13   write to non existent pipe
alrm     14   alarm clock
term     15   software termination signal
urg      16   urgent condition on I/O channel
stop     17   stop signal (not from tty)
tstp     18   stop from tty
cont     19   continue
chld     20   to parent on child exit/stop
gttin    21   to readers pgrp upon background tty read
gttou    22   like TTIN for output if (tp->t_local&LTOSTOP)
io       23   input/output possible signal
xcpu     24   exceeded CPU time limit
xfsz     25   exceeded file size limit
vtalrm   26   virtual time alarm
prof     27   profiling time alarm
winch    28   window changed
lost     29   resource lost (eg, record-lock lost)
usr1     30   user defined signal 1
usr2     31   user defined signal 2
```

Note that cfengine will not attempt to signal or restart processes 0 to 3 on any system since such an attempt could bring down the system. The only exception is that the hangup (hup) signal may be sent to process 1 (init) which normally forces init to reread its terminal configuration files.

`restart "`*`shell command`*`"`

Note the syntax: there is no equals sign here. If the keyword 'restart' appears, then the next quoted string is interpreted as a shell command which is to be executed after any signals have been sent. This command is only issued if the number of processes matching the specified regular expression is zero, or if the signal sent was signal 9 (sigkill) or 15 (sigterm) , i.e. the normal termination signals. This could be used to restart a daemon for instance. Cfengine executes this command and *waits* for its completion so you should normally only use this feature to execute non-blocking commands, such as daemons which dissociate themselves from the I/O stream and place themselves in the background. Some unices leave a hanging pipe on restart (they never manage to detect the end of file condition). This occurs on POSIX.1 and SVR4 popen calls which use wait4. For some reason they fail to find and end-of-file for an exiting child process and go into a deadlock trying to read from an already dead process. This leaves a zombie behind (the parent daemon process which forked and was supposed to exit) though the child continues. A way around this is to use a wrapper script which prints the line "cfengine-die" to STDOUT after restarting the process.

This causes cfengine to close the pipe forcibly and continue. Cfengine places a timeout on the restart process and attempts to clean up zombies, but you should be aware of this possibility.

`owner=,group=`
: Sets the process uid and gid (setuid,gid) for processes which are restarted. This applies only to cfengine run by root.

`chroot`
: Changes the process root directory of the restarted process, creating a ‘sandbox’ which the process cannot escape from. Best used together with a change of owner, since a root process can break out of such a confinement in principle.

`chdir`
: Change the current working directory of the restarted process.

`useshell=`*`true/false/dumb`*
: When restarting processes, cfengine normally uses a shell to interpret and execute the restart command. This has inherent security problems associated with it. If you set this option to false, cfengine executes restart commands without using a shell. This is recommended, but it does mean that you cannot use any shell operators or features in the restart command-line.

  Some programs (like cron) do not handle I/O properly when they fork their daemon parts, this causes a zombie process and normally hangs cfengine. By choosing the value ‘dumb’ for this, cfengine ignores all output from a program and does not use a startup shell. This prevents programs like cron from hanging cfengine.

`matches=`*`number`*
: This option may be used to set a maximum, minimum or exact number of matches. If cfengine doesn't find a number of matches to the regular expression which is in accordance with this value it signals a warning. The ‘<’, ‘>’ symbols are used to specify upper and lower limits. For example,

```
matches=<6  # warn number of matches is greater than or equal to 6
matches=1   # warn if not exactly 1 matching process
matches=>2  # warn if there are less than or equal to 2 matching processes
```

`include=`*`literal`*
: Items listed as includes provide an extra level of selection after the regular expression matches have been expanded. If you include one include option, then only lines containing one or more of the literal strings or wildcards will be matched.

`exclude=`*`literal`*
: Process lines containing literal strings or wildcards in exclude statements are not matched. Excludes are processed after regular expression matching and after includes.

`define=`*`classlist`*
: The colon, comma or dot separated list of classes becomes activated if the number of regular expression matches is non-zero.

`elsedefine=`*`classlist`*
: The colon, comma or dot separated list of classes becomes activated if the number of regular expression matches is zero.

`action=`*`signal/do/warn/bymatch`*
: The default value of this option is to silently send a signal (if one was defined using the `signal` option) to matching processes. This is equivalent to setting the value of this parameter to '`signal`' or '`do`'. If you set this option to '`warn`', cfengine sends no signal, but prints a message detailing the processes which match the regular expression. If the option is set to `bymatch`, then signals are only sent to the processes if the matches criteria fail.

Here is an example script which sends the hang-up signal to cron, forcing it to reread its crontab files:

```
processes:

   "cron" signal=hup
```

Here is a second example which may be used to restart the nameservice on a Solaris system:

```
processes:

   solaris::

       "named" signal=kill restart "/usr/sbin/in.named"
```

A more complex match could be used to look for processes belonging to a particular user. Here is a script which kills ftp related processes belonging to a particular user who is known to spend the whole day FTP-ing files:

```
control:

    actionsequence = ( processes )

  #
  # Set a kill signal here for convenience
  #

    sig = ( kill )

  #
  # Better not find that dumpster here!
  #

    matches = ( 1 )

processes:

   #
   #  Look for Johnny Mnemonic trying to dump his head, user = jmnemon
   #

   ".*jmnemon.*ftp.*" signal=$(sig) matches=<$(matches) action=$(do)

   # No mercy!
```

The regular expression '`.*`' matches any number of characters, so this command searches for a line containing both the username and something to do with ftp and sends these processes the kill signal.

You can arrange for signals to be sent, only if the number of matches fails the test. The `action=bymatch` option is used for this. For instance, to kill process 'XXX' only if the number of matches is greater than 20, one would write:

```
processes:

"XXX" matches=<20  action=bymatch signal=kill
```

See also filters See Section 4.18 [filters], page 92, for more complex searches.

4.30 packages

The packages action allows you to check for the existance of packages on the system, as determined by the package database you select. Optionally, if a package install command was specified, the package can be installed if it is not there.

This operation is set up such that it tries not to make assumptions about the package manager in use. For example, it should be possible to use RPM on a Solaris box.

The syntax summary is:

```
packages:

   class::

      package-name
                          pkgmgr=none/rpm/dpkg/sun
                          cmp=eq/lt/gt/ge/le/ne
                          version=version-string
                          define=class-list(,:.)
                          elsedefine=class-list(,:.)
                          action=none/install/remove

                          ifelapsed=mins
                          expireafter=mins
```

`cmp` Determines how the version of the installed package will be compared to that specified by the `version` attribute. Possible values include:

- `eq` The version installed must be equal to `version`
- `lt` The version installed must be less than `version`
- `gt` The version installed must be greater than `version`
- `le` The version installed must be less than or equal to `version`
- `ge` The version installed must be creater than or equal to `version`
- `ne` The version installed must not be equal to `version`

The default value for this attribute is `eq`.

`version` Specifies the package manager specific version string to match. If this is not specified, then any version matches, and the value of the `cmp` attribute is ignored. See the allowed values of `pkgmgr` below for an explation of how each package manager will interpret this.

`pkgmgr` Selects the package manager database to query. This defaults to either the value of the `DefaultPkgMgr` variable, or if that is not set, there is no default. In that case, no checking will be done unless `pkgmgr` is set explicitly for each package. Note that the default value "`none`" listed is merely a pseudo-value, and cannot actually be used, since it would make no sense anyway.

Each package manager will interpret the `version` and `cmp` attributes in its own way. So, for example, when you use `pkgmgr=rpm`, the comparison will be done with the same rules that RPM use if it were not being run through cfengine.

Currently, the following values are accepted:

`rpm`
: This uses the rpm command, which cfengine expects to find as '`/bin/rpm`' to query the machine's RPM database. The `rpm` check assumes that you are using a version of RPM that understands the concept of an epoch, which means that you will want to use RPM version 3.0.3 or greater. Versions as early as 2.5.6 may work, but it is doubtful. If multiple packages of the same name are installed (i.e. kernel), then the check considers the package to be installed at the specified version if **at least one** of them satisifies the criteria specified by `cmp` and `version`.

  The format of a RPM version string is: [epoch:]version[-release]. The version[-release] can be seen by simply running: `rpm -q <pkg>`. In order to see the epoch, you must use a query format, like this: `rpm -q --queryformat "%{EPOCH}:%{VERSION}-%{RELEASE}\n" <pkg>`. Most packages do not have an epoch, and will print '`(none)`' in the epoch space. In recent incarnations of RPM, the absense of an epoch is interpreted as 0. This is also how cfengine will interpret it. *Be careful with this. If the installed version of a package has an epoch greater than 0, and you do not specify the epoch, unexpected results may happen.* For example, if you have a package installed, '`foo-1:2.0-1`', and you specify a `version=3.0-1` and a `cmp=gt`, the check will be true, because the installed version has an epoch of 1, and you did not specify an epoch, which implies you wanted an epoch of 0. The rule here is basically to always check the epoch of the package you really want, and specify it. It may take a few extra extra seconds to check, but it will save you lots of headaches later.

`portage`
: This uses the emerge command to install packages on a Gentoo Linux system. The 'portage' Python module is used to query the system and find installed packages. The *package_name* and *version* support any package that conforms to the Gentoo package naming specification `http://www.gentoo.org/proj/en/devrel/handbook/handbook.xml?part=2&chap=1#doc_chap2_sect2`, and even most of those in the Portage tree that do not. Because Gentoo package atoms are flexible, you can use the name of the package, along with a *version* and the *cmp* cfengine logic, or you can put the comparison operator and version in the *package_name* field and leave out the *cmp* operator. Because cfengine builds the install string out of the package name, and ignores the *version* you were comparing against, using a package atom with a version is the only way to have a specific version emerged.

```
packages:
```

```
any::

 # Using cmp, if the version of tar is less than
 # 1.16-r2, then the *latest* version of tar will be
 # installed:

 app-arch/tar cmp=ge version=1.16-r2 action=install

 # This is equivalent, but uses the package atom to
 # define the version comparison:

 >=app-arch/tar-1.16-r2 action=install

 # Use a Portage package atom that includes the
 # version.  If version 4.1.22-r1 is not installed,
 # that *exact* version will be installed:

 =dev-db/mysql-4.1.22-r1 action=install
```

`dpkg`
: Please document me!

`sun`
: Please document me!

`freebsd`
: Jo Rhett who implemented this writes: It works like this:

```
control:
  DefaultPkgMgr = ( freebsd )
  FreeBSDInstallCommand = ( "/var/cfengine/packages/pkginstall %s" )

packages:
  bsdsar-1.10_2 action=install
                version=1.10_2
                cmp=ge
```

Note that specifying the version of the package that is part of the freebsd package filename is required. However, it does not use this for comparison, just for the package to install. So you could install 1.10 only if nothing larger than 1.0 was installed.

```
packages:
  bsdsar-1.10_2 action=install
                version=1.0
                cmp=ge
```

You can also backgrade if you know that 1.2 is bad for instance.

```
packages:
  bsdsar-1.10_2 action=install
                version=1.1
                cmp=gt
```

Now, due to the unfortunate fact that freebsd doesn't provide a command line option – only an environment variable – for specifying the source repository, you have a few options.

Get it from the main FreeBSD binary package home:

```
FreeBSDInstallCommand = ( "pkg_add -r %s" )
```

Or write a script that tells pkg_add where to find it, like so:

```
FreeBSDInstallCommand = ( "/var/cfengine/packages/pkginstall %s" )
```

such as

```
#!/bin/sh
PACKAGESITE=http://my.local.repository/packages/
export PACKAGESITE
/usr/sbin/pkg_add -r $*
```

or even

```
#!/bin/sh
cd /my/local/repository
for pkg in $*
/usr/sbin/pkg_add ${pkg}.tbz
```

`define` Specifies the list of classes to define if the specified package is installed.

`elsedefine`
Specifies the list of classes to define if the specified package is not installed.

`action` Specifies whether the packages should actually do anything about the situation it finds. The default for this is to do nothing. Of course, the classes in `define` and `elsedefine` will alwass be defined, as applicable, regardless of the action specified.

`install` Installs the package using the command associated with the selected package manager, if it is not currently on the system at the requested version, as follows:

- RPM - RPMInstallCommand
- DPKG - DPKGInstallCommand
- SUN - SUNInstallCommand

Each variable is of the format:

```
FOOInstallCommand = ( "/usr/bin/foo --args %s --more-args" )
```

The —args are of course optional. The %s is replaced with a space-separated list of the package names that were checked, and found to not be installed.

`remove` CURRENTLY PARSED BUT NOT IMPLEMENTED

NOTE: classes are defined according to the result of the check, not any action performed as a result of that check. In otherwords, if for example you have a situation where a package is not installed, and the action= is set to install, the classes in `elsedefine` will be defined **regardless** of whether or not the install was successful. Assuming the package installed, the next run of cfagent will pick up that fact. This has to be done since the package installs are batched, so there is no reliable way to know if a given package was installed.

Examples:

```
packages:

    redhat_8_0::
```

```
        m4  version=0:1.4.1-11 cmp=eq pkgmgr=rpm elsedefine=needsm4
```

In this first example, we are looking for the m4 package at exactly version 0:1.4.1-11. The installed m4 package on a redhat_8_0 box has no epoch which is the same as zero. Specifying it will keep you out of trouble. This check will cause `needsm4` to be defined if the exact version of m4 specified is not installed.

```
control:
    redhat:;
        DefaultPkgMgr = ( rpm )

packages:
    redhat_8_0::
        make version=0:4.5-2 cmp=ge define=hasmake elsedefine=needsmake
```

In the second example, we use the DefaultPkgMgr variable to set the default for the `pkgmgr` attribute to `rpm`. The actual version of make installed on recent redhat_8_0 machine is 1:3.79.1-14. Since the check is for greater than or equal to this version, the hasmake class will be defined.

```
control:
    redhat:;
        DefaultPkgMgr = ( rpm )
        RPMInstallCommand = ( "/usr/sbin/up2date %s" )

packages:
    redhat_8_0::
        make define=hasmake elsedefine=needsmake action=install
```

This example is much like the second example, except that if the package is not installed, cfengine will attempt to install it using the command in RPMInstallCommand, replacing the %s with the package name, `make`. If there were multiple packages specified in this way, the package installation would occur at the end of the package checks, and one command would be run, with %s replaced with a list of all package names. In this example we chose not to use a version spec, but it is allowed, and as always, is optional.

NOTE here that if make was not installed when the check is made, `needsmake` is defined, regardless of whether or not the install succeeds. If the install is successful, the next cfagent run will define `hasmake`.

4.31 rename

As of version 2.1.0 rename is a synonym for disable, See Section 4.15 [disable], page 70.

4.32 required

This is a synonym for disks, See Section 4.13 [disks], page 67. This action tests for the existence of a file or filesystem. It should be called after all NFS filesystems have been mounted. You may use the special variable `$(binserver)` here.

```
required:

  /filesystem freespace=size-limit define=class-list(,:.)

   ifelapsed=mins expireafter=mins
```

Files or filesystems which you consider to be essential to the operation of the system can be declared as 'required'. Cfengine will warn if such files are not found, or if they look funny.

Suppose you mount your filesystem `/usr/local` via NFS from some binary server. You might want to check that this filesystem is not empty! This might occur if the filesystem was actually *not* mounted as expected, but failed for some reason. It is therefore not enough to check whether the directory `/usr/local` exists, one must also check whether it contains anything sensible.

Cfengine uses two variables: `sensiblesize` and `sensiblecount` to figure out whether a file or filesystem is sensible or not. You can change the default values of these variables (which are 1000 and 2 respectively) in the `control` section. See Section 4.9 [control], page 33.

If a file is smaller than `sensiblesize` or does not exist, it fails the 'required' test. If a directory does not exist, or contains fewer than `sensiblecount` files, then it also fails the test and a warning is issued.

```
required:

   any::

      /$(site)/$(binserver)/local
```

If you set the `freespace` variable to a value (the default units are kilobytes, but you may specify bytes or megabytes), e.g.

```
required:

  /site/host/home1 freespace=50mb define=dotidy
  /site/host/home2 freespace=10% define=dotidy
```

then cfengine will warn when the filesystem concerned has less than this amount of free space. By adding a `define` tag, you can switch on any number of classes if this happens. This allows you to activate special measures for dealing with a filesystem which is in danger of becoming full.

4.33 resolve

The file `/etc/resolv.conf` specifies the default nameserver for each host, as well as the local domain name. This file can also contain other information, but these are the only two things cfengine currently cares about. In specifying nameservers you should use the dotted numerical form of the IP addresses since your system may not understand the text form if it is not correctly configured. You may list as many nameservers as you wish, with the default server at the top of the list. The resolver normally ignores entries if you add more than three. The statement:

```
resolve:

  mygroup::

     129.240.22.35
     129.240.22.222
     129.240.2.3
```

declares a list of nameservers for hosts in the group or class `mygroup`. When you add the `resolve` command to the `actionsequence`, this declaration together with the `domain` variable (set here to `uio.no`) results in a `/etc/resolv.conf` file of the form:

```
domain uio.no
nameserver 129.240.22.35
nameserver 129.240.22.222
nameserver 129.240.2.3
```

Note that the `resolve` action does not delete anything from the file `/etc/resolv.conf` unless the `EmptyResolvConf` variable is set to 'true' . It adds nameservers which do not previously exist and reorders the lines of servers which do exist.

As of version 1.3.11, you may use a quoted string to add non-nameserver lines to this file. For example:

```
resolve:

  mygroup::

     129.240.22.35
     129.240.22.222
     "# Comment line"
     "order bind, files"
```

If the line begins with a non-numeric character, the word 'nameserver' is not added to the line.

4.34 scli

SCLI is an SNMP command line interpreter written by IETF leader Juergen Schoenwaelder of the Jacobs University of Bremen. It is a command interpreter used to simplify the interaction with SNMP enabled devices.

This cfengine interface is a special case of the `shellcommands` interface, which passes commands to SCLI instead of to '`/bin/sh`'. The command options are the same as those below for `shellcommands`.

```
shellcommands:

  class::

    "scli-string"  shellcommand options
    "scli-string"
```

This integration was funded by the IST-EMANICS Network of Excellence in Network and Service Management, of which Oslo University College and the Jacobs University of Bremen are members.

4.35 shellcommands

Cfengine focuses on fairly simple-minded tasks in order to be as general as possible. In many cases, you will therefore want to write a script to do something special on your system. You can still take advantage of the classes you have defined by executing these scripts or shell commands from this section.

The syntax is simply to quote the command you wish to be executed.

```
shellcommands:

  class::

    "command-string"

          timeout=seconds
          useshell=true/false
          umask=octal number
          owner=uid
          group=gid

          background=false/true
          chdir=directory
          chroot=directory
          preview=true/false
          inform=false/true

          noabspath=false/true

          ifelapsed=mins
          expireafter=mins

          define=class-list
          elsedefine=class-list
```

command-string
: This is the command to be executed.

timeout
: If you set the optional *timeout* parameter, then cfengine will abort the specified shellcommand if it exceeds the given time-limit (specified in seconds). This can be useful for avoiding hung programs caused by hung network connections, etc. Timeouts are generated by alarm interrupts within a single agent. This can be contrasted with `expireafter` in which a second agent is required to interrupt a command.

useshell
: Some program lines, especially those that do not use any shell-specific capabilities (such as redirection and wildcard expansion) can be run without the shell. This is typically more secure, as the command line is not altered by the user or by the system. It is also faster, as the shell does not have to be spawned in order to run the given command. Use the *useshell* parameter to tell cfengine to not use the shell to run this shellcommand.

umask
: The umask affects the permissions given to a file created by this shellcommand. The umask specifies, specifically, the permissions that are to be taken away.

owner

group
: The user and group ID's of the process can be set (using the *owner* and *group* parameters respectively) to restrict the permissions of the shellcommand. This can only be done if cfengine is executed by root; otherwise, the user and group will remain that of the the user who started cfengine.

background
: Run this command in the background if this is specified true. This will make cfengine run faster, but no tests can be made (at least directly) on the results of this command. Also when jobs are backgrounded, the output in verbose mode can be somewhat erratic due to the unsychnronized streams.

chdir
: Change to the specified directory before running this command.

chroot
: The `chroot` option changes the process root directory of the command, creating a 'sandbox' which the process cannot escape from. Best used together with a change of owner (using the `owner` parameter), since a root process can break out of a chrooted environment.

preview
: The `preview` option means that the shellcommand will also be executed during the `--dry-run` (`-n`) options. This allows cfengine to be more aware of the results of scripts which define classes. This option should be used with care.

  Scripts should conform to the protocol of not executing unnecessary commands when the classes `opt_dry_run` is defined.

inform

noabspath
: Normally, cfagent requires the command string to begin with a '`/`' since it is dangerous to rely on an implicit path. However, sometimes it it is appropriate to override this. This behavior can be overridden using the *noabspath* parameter.

ifelapsed
: The shellcommand specified will not be run unless the specified amount of time (in minutes) has elapsed since the command was previously run.

expireafter
: If this amount of time (in minutes) has elapsed since the command started, then the command is aborted by a second agent that is patrolling the system.

define
: Define the specified classes if the command finishes successfully.

elsedefine
: Define the specified classes if the command does not finish successfully.

Variable substitution works within the strings. Here are some examples.

```
shellcommands:

   sun4::

       "/usr/lib/find/updatedb"

   AllHomeServers.Sunday::

       "/dir/noseyparker /$(site)/$(host)/u1 $(sysadm) nomail"

   AllBinaryServers.sun4.Saturday::

      "/usr/etc/catman -w -M /usr/local/man"
```

```
    "/usr/etc/catman -w -M /usr/local/X11R5/man"
    "/usr/etc/catman -w -M /usr/man"
    "/usr/etc/catman -w -M /usr/local/gnu/man"
```

If you need to write more advanced scripts which make detailed use of the classes defined by cfengine, use the `$(allclasses)` variable to send a complete list of classes to your script. An enviornment variable, CFALLCLASSES, is set and is in the format

```
CFALLCLASSES=class1:class2:class3...
```

This variable is kept up-to-date at any given time with only the classes which are defined. The command line option '`-u`' or '`--use-env`' can be used to define an environment variable which will be inherited by all scripts and contains the same information. This is not the standard approach, since some systems cannot cope with this rapid change of environment and generate a Bus Error.

Commands can be iterated over variable lists, provided there is at least one space between each variable. For example:

```
control:

      actionsequence =
         (
         shellcommands
         )

 var1 = ( a:b:c )
 var2 = ( x:y:z )

shellcommands:

   "/bin/echo $(var1) $(var2)"
```

This iterates over all values of the list variables. See ⟨undefined⟩ [Iterating over lists], page ⟨undefined⟩. If you are iterating over a list, the time limit (in seconds) which is specified in the *timeout* parameter applies to each separate iteration, not to the sum total of all the iterations.

4.36 strategies

Strategies (introduced in cfengine version 2.0) are a way of picking from a set of classes randomly. Each class is a possible course of action. A strategy group (of classes) is defined as follows:

```
strategies:

  { my_strategy_alias

  class1: "2"
  class2: "3"
  class3: "$(value)"
  class4: "6"
  class5: "1"
  }

tidy:

 class1.Hr00::

   /home pat=*.mp3 age=0

 class2.Hr02::

   /home pat=*.wav age=0
```

The idea here is to randomly pick from a selected set of classes.

Specifically, each strategy is a class which is defined with a certain probability. An integer weight is provided in quotes to represent the probability weight of the associated class. When cfengine is run, it randomly picks one of the classes from each strategy. Using strategies, you can choose different ways of configuring or protecting a system, at random, thus confounding environmental attempts to break into the system.

Note that each strategy has a formal name (such as '`my_strategy_alias`' in the example), but this name is not used to attach a strategy to an action the same way that filters or ACLs are.

4.37 tidy

The tidy function is used to delete (remove permanently) unwanted files from a system. It is useful for tidying up in `/tmp` or cleaning out `core` files from users' home directories. The form of an entry is:

```
tidy:

  class::

     /directory
                     pattern/include=wildcard

                     recurse=number/inf
                     age=days
                     size=number/empty
                     type=ctime/mtime/atime
                     dirlinks=keep/tidy/delete
                     rmdirs=[true/all]/[false/none]/sub
                     links=stop/keep/traverse/tidy

                     define=classlist
                     elsedefine=classlist

                     syslog=true/on/false/off
                     inform=true/on/false/off
                     ifelapsed=mins
                     expireafter=mins

                     filter=filter alias
                     ignore=pattern
                     exclude=pattern
                     xdev=true/on/false/off
```

Note that, each of the options below can be written in either upper or lower case and abbreviated by any unique abbreviation.

/directory
: This is the directory name to directories which mark the start of a search for files matching certain wildcards. The wildcard `home` may be used instead of an explicit directory, in which case cfengine iterates over all home directories. It is compulsory to specify a directory.

`pattern=`*wildcard* `or include=`*wildcard*
: A wildcard or filename to match the files you wish to be deleted. The pattern may contain the special symbols '`?`' which matches a single character and '`*`' which matches any number of characters as in the shell. These two options are synonymous, as of version 2.0.x. Note that, this pattern is processed as a filter before any other filter and, for safety reasons, it defaults to nothing. Thus, if you want to use a filter to select the files, you should set '`pattern=*`', else the filter will not see any files at all.

`exclude=`*`wildcard`*
: This does not work for the home directive; use the global ignore list for this.

`ignore=`*`wildcard`*
: This does not work for the home directive; use the global ignore list for this.

`recurse=`*`number/inf`*
: This specifier tells cfengine whether or not to recurse into subdirectories. If the value is zero, only the named file or directory is affected. If the value is 1, it will open at most one level of subdirectory and affect the files within this scope. If the value is `inf` then cfengine opens all subdirectories and files beginning from the specified filename. See Section 4.17.2 [Recursion], page 89.

`age=`*`days`*
: The age of a file in days represents a minimum *access* time elapsed before the file will be deleted. In other word a file will be deleted if it has not been accessed for *days* days.

`links=`*`stop/traverse/tidy`*
: Normally cfengine does not descend into subdirectories which are pointed to by symbolic links. If you wish to force it to do so (without using the `-l` command line option) you may give this option the value `true`, or `traverse`, or `follow`. To specify no recursion you set the value `false` or `stop`. Note that the value set here in the cfengine program *always overrides* the value set by the `-l` command line option, so you can protect certain actions from this command line option by specifying a negative value here. If you specify no value here, the behaviour is determined by what you specify on the command line.

  The value `links=tidy` has the same effect as the '`-L`' command line option except that here it may be specified per item rather than globally. Setting this value causes links which point to non-existent files to be deleted. This feature will not work on commands with the '`home`' wildcard feature. If you want to clean up old links you should either user a `files` command or the command line option which sets the tidy feature globally.

`size=>`*`number/empty`*
: Old syntax `size=`*`number/empty`*. The value of this parameter decides the size of files to be deleted. Files larger than this value will be deleted if they also are older than the time specified in `age`. The default size is zero so that any file which gets matched by another critereon is deleted. However, if you want to single out only totally empty files, the `empty` may be used. With this option only empty files, nevery files with anything in them, will be deleted, if older than `age`. By default, the filesizes are in kilobytes, but kilobytes and megabytes may also be specified by appending b,k,m to the numbers. Only the first character after the number is significant so you may write the numbers however it might be convenient, e.g. *`14k`*, *`14kB`*, *`14kilobytes`*, the same as for `disable`.

`type=`*`ctime/mtime/atime`*
: This value is used to set the type of time comparison made using `age`. The default is to compare access times (atime) or the last time the file was read. A comparison by modification time (mtime) uses the last time the contents of the file was changed. The ctime parameter is the last time the contents, owner or

permissions of the file were changed. Note that on directories, mtime is always used for comparisons, since the very act of stat'ing alters atime and makes this comparison meaningless.

`dirlinks=`*`keep/tidy/delete`*
: This value is used to decide whether cfengine will delete links which point to directories. The default value is to keep the links. Note that, if the `travlinks` option is switched on, cfengine will not tidy or delete links which point to directories, instead it follows them into the subdirectory. This is a supplement to the `rmdirs` option. You need both to make links to directories disappear. Note that, even if `travlinks` is set to true, cfagent will not follow symbolic links that are not owned by the agent user ID; this is to prevent link race attacks, in which users with write access could divert the agent to another part of the filesystem,

`rmdirs=`*`true/false/all/sub`*
: Normally cfengine will not delete directories. If this option is set to 'true' then cfengine will delete any directories which are *empty*. Non-empty directories will not be touched and no message will be given unless in verbose mode. Note that this option overrides the above option `dirlinks`, so that even links which point to empty directories will be removed. If this is set to 'sub' then the topmost directory will not be removed, only sub-directories.

`define=`*`classlist`*
: The colon, comma or dot separated list of classes becomes defined if any file matching the specified pattern is deleted.

`xdev`
: Prevents cfengine from descending into file systems that are not on the same device as the root of the rescurion path.

Take a look at the following example:

```
tidy:

   AllHomeServers::

       home     pattern=core   R=inf age=0
       home     pattern=*~     R=inf age=7
       home     pattern=#*     R=inf age=30


   any::

       /tmp/    pat=*            R=inf   age=1
       /        pat=core         R=2     age=0
       /etc     pat=hosts.equiv  r=0     age=0
```

In the first example, all hosts in the group `AllHomeServers` iterate a search over all user home directories looking for 'core' files (older than zero days) and `emacs` backup files '`*~`', '`#*`' older than seven days.

The default values for these options are the empty string for the wildcard pattern, zero for the recursion and a specification of the age is compulsory.

When cfengine tidies users' home directories, it keeps a log of all the files it deletes each time it is run. This means that, in case of accidents, the user can see that the file has been deleted and restore it from backup. The log file is called `.cfengine.rm` and it is placed in the home directory of each user. The file is owned by root, but is readable to the user concerned.

4.38 unmount

The unmount function unmounts non-required filesystems and removes the appropriate entry from the filesystem table (`/etc/fstab` or equivalent). The syntax is simply

```
unmount:

   class::

      mounthost:filesystem

        deletedir=true/false
        deletefstab=true/false
        force=true/false
        ifelapsed=mins
        expireafter=mins
```

The options allow you to temporarily unmount a directory without actually removing it from the filesystem table. The option `force` is not currently implemented and will likely have to be system dependent. For example:

```
unmount:

   physics::

      libraryserver:/$(site)/libraryserver/data
```

If the device is busy then the actual unmount will not take place until it becomes free, or the machine is rebooted. This feature should work on AIX systems, in spite of these machines inherent peculiarities in the form of the filesystem table.

Some users do not mount filesystems on a directory of the same name as the source directory. This can lead to confusion. Note, if you have problems removing a mounted filesystem, try using the mountpoint of the filesystem, rather than the name of the filesystem itself, in the unmount command.

5 Cfservd and cfrun reference

The server daemon is controlled by a file called '`cfservd.conf`'. The syntax of this configuration file is deliberately modelled on cfengine's own configuration file, but despite the similarities, they are separate.

You can use `groups` and `import` in both files to break up files into convenient modules and to import common resources, such as lists of groups.

Note that the classes in the '`cfservd.conf`' file do not tell you the classes of host which have access to files and directories, but rather which classes of host pay attention to the access and deny commands when the file is parsed.

Authentication is not by class or group but by hostname, like the '`/etc/exports`' file on most Unix systems. The syntax for the file is as follows:

```
control:

  classes::

      domain = ( DNS-domain-name )

      cfrunCommand = ( "script/filename" )  # Quoted

      MaxConnections = ( maximum number of forked daemons )

      # ChecksumDatabase = ( filename ) - deprecated

      IfElapsed = ( time-in-minutes )

      DenyBadClocks = ( false )

      AllowConnectionsFrom = ( IP numbers )

      DenyConnectionsFrom = ( IP numbers )

      AllMultipleConnectionsFrom = ( IP numbers )

      TrustKeysFrom  = ( IP numbers )

      AllowUsers = ( mark systemuser )

      LogAllConnections = ( false/true )

      LogEncryptedTransfers = ( false/true )

      SkipVerify = ( IP numbers )

      DynamicAddresses = ( IP numbers )

      BindToInterface = ( IP number/hostname )

      HostnameKeys = ( true/false )

groups:

  Group definitions

import:

  Files to import

admit: | grant:

  classes::

     /file-or-directory

       wildcards/hostnames

deny:

  classes::

     /file-or-directory

       wildcards/hostnames root=hostlist encrypt=true/on
```

Iteration of variables is allowed, hence:

```
control:
  Split = ( " " )
  hostlist = ( "10.10.10.1 10.10.10.2 10.10.10.3" )
  dirs =  ( "bin etc lib" )
  base = ( /usr )

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

admit:
   $(base)/$(dirs)   $(hostlist)
```

results in:

```
Path: /usr/bin (encrypt=0)
   Admit: 10.10.10.1 10.10.10.2 10.10.10.3 root=
Path: /usr/etc (encrypt=0)
   Admit: 10.10.10.1 10.10.10.2 10.10.10.3 root=
Path: /usr/lib (encrypt=0)
   Admit: 10.10.10.1 10.10.10.2 10.10.10.3 root=
```

The file consists of a control section and access information.

5.1 control

5.1.1 IP address ranges

In the access control lists below, host ranges can be specified in a number of ways i) as substrings, ii) as address ranges denoted by the "-" hyphen, or iii) as CIDR (Classless Inter Domain Routing) notation. For example

```
128.39.73
128.39.74.10/23
128.39.74-75.10-22
2001:700:700:3:290:27ff:fea2:4730-4790
2001:700:700:3:290:27ff:fea2:4730/64
```

In the CIDR notation, the slash followed by a number indicates the netmask, or the number of bits which are common to a group of hosts. Normally, this is connected to a specific subnet, but here it simply represents the number of bits from the left which are fixed for matching; all remaining bits are wildcards. The following forms are equivalent:

```
128.39.74.
128.39.74.10/24
128.39.74.1-254
```

5.1.2 AllowConnectionsFrom

This variable allows a list of numerical IP masks to be specified, which cfservd will allow connections from. If the list is not empty and a host whose IP address is not specified attempts to connect to the daemon, its connection will be closed immediately. This can be used to prevent hanging connection attacks from malicous hosts and other denial of service attacks which would bind thread resources.

```
control:

 AllowConnectionsFrom = ( 128.39.89  192.2.0.10 )
```

5.1.3 AllowMultipleConnectionsFrom

This variable should contain a list of IP wildcards to hosts which are allowed simultaneous sessions on the server. Hosts which are not in this list are allowed to connect only once, i.e. they must terminate and reconnect in order to establish a new session. This is to prevent a possible attacker from opening multiple sockets and never closing them, resulting in a denial of service attack. Hosts IP's can be placed here if they could have overlapping copy sessions (e.g. long backup transfers which can run over time). This prevents the error message "Multiple connections denied/spam shield".

5.1.4 AllowUsers

This list determines which users are to be allowed to connect to the daemon. Note that there is no way of identifying users except by their public keys. If a malicious asserts their identity, when no public key for the named user is known to the server, then they could spoof the identity of that user. All users who should be allowed to connect need to be here. This applies to use of cfrun.

```
AllowUsers = ( mark root )
```

In other words, this is a "security by obscurity" first defence against picking up bad keys, when the server is in trust mode, with respect to a host. The attacker must know a valid user name in order to even try their luck entering into a key dialogue.

This reduces the probability that spoofing can be successful. The only real defence against spoofing is to make sure that all required public keys are installed in advance, and to switch off trust.

5.1.5 AutoExecCommand

This variable no longer exists in cfengine version 2.

5.1.6 AutoExecInterval

This variable no longer exists in version 2 of cfengine.

5.1.7 BindToInterface

If this is set to a specific IP address of an IP configured interface, cfservd will listen for connections only on that interface. On Multi-homed hosts this allows one to restrict the traffic to one interface. Note, Unix only allows one or all interfaces to be selected. An interface must be configured with an IP address in order to be bound.

5.1.8 ChecksumDatabase

(This variable is deprecated as of version 2.1.21. The location of the database will be fixed internally.)

This is the path and filename to a database which will cache MD5 checksum values server-side. This optimization is only available if you have the Berkeley database library '`libdb`' on your system. If this variable is not defined, no database caching will be used and checksum values will be computed directly on request. The utility of this solution is a trade-off between the time it takes to compute the checksum versus the time for a disk-based lookup.

5.1.9 cfrunCommand

This string is the command which you would like to be executed remotely by the `cfrun` command.

5.1.10 DenyBadClocks

If this is set to `off`, cfservd will not deny access to clients whose clocks are off by more than one hour. The default is to deny access to systems whose clocks differ by more than one hour. This can prevent messages of the form 'Can't stat' file when remote copying.

5.1.11 DenyConnectionsFrom

Hosts which are included by the allow-list above can be explicitly denied access using this list.

```
control:

 DenyConnectionsFrom = ( 128.39.89.76 )  # rogue host
```

5.1.12 HostnameKeys

If this variable is set to true/on, it causes cfservd to lookup and store trusted public keys according to their DNS fully qualified host name, instead of using the IP address. This can be useful in environments where hosts do not have fixed IP addresses, but do have fixed hostnames.

```
HostnameKeys = ( on )
```

This method of storing keys is not recommended for sites with fixed IP addresses, since it removes one security barrier from a potential attacker by potentially allowing DNS spoofing.

5.1.13 IfElapsed

The `IfElapsed` anti-spamming filter is also built into `cfservd` so that a remote user cannot even get as far as causing cfengine to parse its input files (which could be used for spamming in itself). The time is in minutes, the default is one hour.

5.1.14 LogAllConnections

If set to true, every successful connection will be logged to syslog. This could be useful for identifying abuses of the service, if the server should come under attack, e.g. a denial of service attack. The IP address can then be excluded from the allowed connections list.

5.1.15 LogEncryptedTransfers

If set to true, every successful request for a file that is granted access only with an encrypted connection is logged in syslog.

5.1.16 MaxConnections

This integer value sets a limit on the maximum number of child daemon threads which cfservd will 'fork' in order to handle remote requests. The default value is ten.

5.1.17 TrustKeysFrom

Hosts which are included in this list are automatically trusted, if cfservd does not know their public key. This allows public keys to be exchanged. Cfservd will not automatically accept a public key from a host it does not know, since the key will be used to assert strong authentication later. Once a public key has been associated with an IP address, it will never be updated, unless the existing key is deleted by hand.

```
control:

 TrustKeysFrom = ( 128.39.89.76 )     # trusted host
 TrustKeysFrom = ( 128.39.89.76/24 )  # trusted subnet
```

5.1.18 DynamicAddresses

Hosts which are included in this list are assumed to have IP addresses which can change with time, e.g. hosts which are given IP addresses by DHCP or a BOOTP like protocol.

```
control:

 DynamicAddresses = ( 128.39.74.100-200 )  # DHCP range
```

If cfservd receives a connection from an IP address that is in this list, and trustkey is *true*, the existing key for that IP address can be replaced with a new key, and the old key is recorded in a "used keys" list, access is granted. If trust is switched off, the server looks in the "used key list" to see if the key has been seen before. If not access is refused. If it has been seen before – it uses this earlier trust to accept the connection and replace the IP-key binding.

Note that used keys are kept in a database for easy lookup, whereas fixed keys are kept in files for easy administration. If host keys change or are reinstalled on the dynamically allocated hosts, then this database should probably be deleted to purge keys that become illegal.

DHCP addresses have no effect on hosts that use IPV6, since IPV6 addresses are generally issued based on an interface MAC address and do not therefore change.

5.2 admit, grant and deny

5.2.1 `root=`

This list specifies the names of hosts which are to have read access to files, regardless of the owner of the file. This effectively gives root users on connecting hosts privileges to non-root owned files on the server, but not vice-versa, similar to the NFS root mapping, except that there is no question of a client being able to modify files on the server. Caution: cfservd trusts the DNS service, so be aware that cache poisoning attacks are a possible way of bypassing access controls.

Cfservd requires you to grant access to files without following any symbolic links. You must grant access to the real file or directory in order to access the file object. This is a security feature in case parties with login access to the server could grant access to additional files by having the permission to create symbolic links in a transitory directory, e.g. '/tmp'.

As of version 2.0.4: Once a verified host address has been identified with a functioning public/private key authentication, the IP address is added to the SkipVerify list, so that time is not wasted in verifying reverse lookups, when the identify can be verified more efficiently and securely by a key mechanism.

5.2.2 encrypt=true

If this option is set, cfservd will only serve the named files if the copy access type is `secure`, i.e. on an encrypted link. This presupposes that cfengine has been compiled with a working OpenSSL library.

5.2.3 SkipVerify

If connecting hosts use a Network Address Translator in order to share an IP address, reverse lookup will fail to give a correct verification of host identity. You can switch off cfservd's verification of IP host identity for specific IP addresses or patterns using this command. E.g.

```
SkipVerify = ( 192.0.0.10  192.0.2.  )
```

This does not affect key verification.

NOTE!! This is a security risk because it means that cfservd implicitly trusts the connecting hosts! You should be very careful in using Network Address Translators in a secure environment. It is not recommended for sites which require a high level of security.

5.3 cfrun

The general syntactic form of the `cfrun` command is

```
cfrun -option --longoption class1 class2 ...
```

Since `cfrun` addresses remote hosts, there is an ambiguity in whether options are intended for the `cfrun` command itself, on the local host, or whether they are to be passed on to the agent on the remote hosts. To clarify this distinction, the arguments are organized as follows:

```
cfrun -local options -- remote options -- remote classes
```

Local options are processed by `cfrun` on the local host; remote options are passed on as options to the remote `cfagent` (actually to the command defined in `cfrunCommand` in the file '`cfservd.conf`'; remote classes are processed by the remote `cfservd` service, and specifiy classes which must be satisfied by the remote host in order to invoke the remote command.

The '-q' and '-I' options are always assumed when executing cfengine remotely, so that SplayTime is effectively zero when polling hosts serially, and the output always shows what is happening on the remote hosts.

On connecting to a remote host, cfengine attempts to obtain credentials by exchanging keys. Unknown keys, in a key exchange, need to be explicitly accepted on trust. Normally, the interactive cfrun program prompts the user explicitly, (like in the secure shell, ssh, connections). This can be annoying if there are many hosts to connect to. The '-T' option tells cfengine to trust all new keys. This option should be used with caution, and only at times when one is sure that the hosts one is connecting to are trustworthy.

Each host evaluates the classes sent by cfrun and decides whether cfengine should be invoked. Only hosts which belong to the classes defined on the cfrun command line are executed. This allows you to single out groups of hosts which should execute cfengine, based on the very classes which you have defined for your configuration. If no classes are sent on the command line, then all hosts are run.

cfrun uses a configuration file which is located under the CFINPUTS directory in order to determine which hosts and in which order it should try to connect. Because cfengine always uses a reliable TCP protocol for connections, it verifies each connection rather than simply broadcasting openly. Using this file you can even simulate broadcasting to hosts outside your subnet.

This file should contain every host name you ever want to configure remotely, because you can still select subsets of the file by specifying classes which the remote host will understand. If the remote host is not in one of the classes you specify when you run cfrun, then it will simply ignore the request. Conversely, if you do not place a host in this file, it will never be contacted when you use the cfrun command. The format of the file is as follows

```
#
# Comment ..
#
domain=my.domain
access=user1,user2
outputdir=directory
maxchild=number limit
hostnamekeys=true/false

hostname1           options
hostname2:port  options
...
include=cfrun.site1.external.hosts
include=cfrun.site1.internal.hosts
include=cfrun.site2.private.hosts
include=cfrun.site2.shared.hosts
```

If the option outputdir is present, cfrun forks a separate process for each host and passes the output to files in a named directory. The maxchild line limits the number of forked processes.

It is important to add the domain-name to this file. The options you specifiy in this file, per host, are added to those you might specify on the command line when invoking cfengine remotely. For instance, you might know of a bug on one host and decide not to perform interface configuration on that one machine. You would write a line like this:

```
funny.domain -- -i  # problem host
```

You could use `cfrun` inside one of your cfengine configuration files in order to remotely execute cfengine on all of the other network machines, by setting up a host list. The disadvantage however is that cfengine has to poll the systems on the network, which means that cfengine cannot be working in parallel on all hosts.

Some other examples:

```
e.g.  cfrun -- -- linux          Run on all linux machines
      cfrun -- -p                Ping and parse on all hosts
      cfrun -v -- -p             Ping all, local verbose
      cfrun -v -- -k -- solaris  Local verbose, all solaris, but no copy
```

Amongst the local options, one may specify a subset of the hosts which are to be contacted by cfrun, i.e. to avoid processing the entire list of hosts. For example, to contact only host1 and host2, given that they are already in the list of hosts.

```
cfrun -v host1 host2
cfrun -v host1 host2 -- -p
```

5.4 Firewalls and NATs

Firewalls and Network Address Translators (NAT) can be a problem for addressing. Suppose you have a firewall and with a private IP-range behind the firewall. You want to update the nodes from a central host. You can do a two stage configuration: first update the firewall and then update from the firewall to the nodes.

But suppose you already use SNAT (Source Network Address Translation) and DNAT (Destination ...) for the nodes. With DNAT you can say that socket 22000 on the firewall is routed to *host-name*:5308. DNAT gives us the possibilty to update the nodes from a central server in one step instead of two.

If the port command is given cfrun uses this to connect to the client instead of the default (5308) one. Here is an example ('`cfrun.hosts`'):

```
domain=example.org
access=mark,sigmund
hostnamekeys=true

node1.example.org
node2.example.org:22000 -DNis
node2.example.org:22001
```

This connects to: 1) node1 with standard port, 2) node2 with port 22000 and extra options -DNis and, 3) node2 with port 22000.

6 Cfexecd reference

In wrapper mode (non-forking, non-daemon mode), cfagent is run by adding a line to the root crontab file of each system:

```
0,30 * * * * /usr/local/sbin/cfexecd -F
```

This is enough to ensure that cfengine will get run. Any output generated by this job, will be stored in '`/var/cfengine/outputs`'.

The program `cfexecd` operates as a wrapper for cfagent. It has the following options:

```
-h (--help)
-d (--debug)
-v (--verbose)
-f (--file)
-q (--no-splay)
-F (--no-fork)
-1 (--once)
-g (--foreground)
-p (--parse-only)
-L (--ld-library-path)
```

In addition, if you add the following to the file '`cfagent.conf`', the system administrator will be emailed a summary of any output:

```
control:

smtpserver    = ( mailhub.example.org ) # site MTA which can talk smtp
sysadm        = ( mark@example.org )   # mail address of sysadm
EmailMaxLines = ( n )            # max lines of output to email
OutputPrefix  = ( "!" )                # Line prefix
```

Fill in suitable values for these variables. `EmailMaxLines` may be set to 0 to disable email output, a postive integer to set a limit, or `inf` to email the whole output regardless of its size. If undefined, `EmailMaxLines` defaults to 100.

An alternative, or additional way to run cfengine, is to run the '`cfexecd`' program is daemon mode (without the '`-F`') option. In this mode, the daemon lives in the background and sleeps, activating only in accordance with a scheduling policy. The default policy is to run once every hour (equivalent to `Min00_05`). Here is how you would modify '`cfagent.conf`' in order to make the daemon execute cfagent every half-hour:

```
control:

 # When should cfexecd in daemon mode wake up the agent?

 schedule   = ( Min00_05 Min30_35 )
```

Note that the time specifications are the basic cfengine *time classes*. Although one of these methods should suffice, no harm will arise from running both cron and the cfexecd side-by-side. Locking mechanisms are used by `cfagent` to ensure that no contention will occur.

Note, that if problems with library path for compiled-in libraries occur, an explicit library path can be specified with the '`-L`' option.

```
0,30 * * * * /usr/local/sbin/cfexecd -F -L /local/iu/lib:/local/lib:/local/gnu/lib
```

The output generated by a cfagent run is collected and stored with date stamps in the '`outputs`' subdirectory of the work directory (usually '`/var/cfengine/outputs`'). If cfengine has a valid smtp server configured it will attempt to E-mail new reports to the system administrator. Duplicate reports are suppressed however for a period of one day. Thus, if one has a repeating message, then it will only be sent by E-mail once per day – this feature is meant to prevent cfagent from spamming administators with multiple, indentical reports. As soon a report different from the previous one is received, the memory is reset.

Note: this repeated message suppression feature cannot work if you include time dependent data in messages, i.e. if you include the date or time in an alert, then clearly the message will be a different message each time. Output from cfengine should not contain the time or date, except in the E-mail header.

7 Problem solving

7.1 'cf.preconf' bootstrap file

In some cases you will want to run cfengine on a system to configure it from scratch. If the system is in a very bad way, it might not even be able to parse the cfengine configuration file, perhaps because the network was not properly configured or the DNS (Domain Name Service) was out of action. To help prevent this situation, cfengine looks for a script called `cf.preconf` which gets executed prior to parsing and can be used to perform any emergency tests. This file needs only contain enough to get the system to parse the configuration files.

`cf.preconf` may be any script in any language. It need not exist at all! It is fed one argument by cfengine, namely the system hard-class for the current system (e.g. `ultrix`). Here is an example:

```
#!/bin/sh
#
# cf.preconf is an emergency/bootstrap file to get things going
# in case cfengine is unable to parse its config file
#

backupdir=/iu/nexus/local/iu/etc

 #
 # If these files don't exist, you might not be able to parse cfagent.conf
 #

if [ ! -s /etc/resolv.conf ]; then

 echo Patching basics resolv.conf file
 cat > /etc/resolv.conf << XX
domain iu.hioslo.no
nameserver 128.39.89.10
XX

fi

#
# SVR4
#

if [ "$1" = "solaris" ]; then

  if [ ! -s "/etc/passwd" ]; then

  echo Patching missing passwd file
    /bin/cp $backupdir/passwd /etc/passwd
  fi

  if [ ! -s "/etc/shadow" ]; then

   echo Patching missing passwd file
   /bin/cp $backupdir/shadow /etc/shadow
  fi
fi
```

```
#
# BSD 4.3
#

if [ "$1" = "linux" ]; then

   if [ ! -s "/etc/passwd"  ]
   then

    echo Patching missing passwd file
    /bin/cp $backupdir/passwd.linux /etc/passwd
   fi
fi
```

Note - in some circumstances, it might be appropriate to exit cfengine altogether after this script. If the script outputs a string containing the text "cfengine-preconf-abort", then cfagent will abort execution immediately after this.

7.2 'cfrc' resource file

If, for some reason you are not satisfied with the defaults which cfengine uses, then you can change them by making an entry in the resource file. The default values are defined in the source code file `classes.c` in the distribution. The format of the resource file is:

```
hardclass.variable: value
```

For example, you might want to forget about where your HPUX system mounts its mail directory and mount it under `/usr/spool/mail`. In this case you would add the line:

```
hpux.maildir: /usr/spool/mail
```

To redefine the filesystem table for GNU/linux, you would write:

```
linux.fstab: /etc/linuxfstab
```

The full list of re-definable resources is:

```
   mountcomm       # command used to mount filesystems
   unmountcomm     # command used to unmount filesystems
   ethernet        # name of the ethernet device
   mountopts       # options to above mount command
   fstab           # the name of the filesystemtable
   maildir         # the location of the mail directory
   netstat         # the full path to netstat and options
   pscomm          # the path to the system's ps command
   psopts          # the options used by ps (default aux/ef)
```

You should never need to redefine resources unless you decide to do something non-standard. Interested readers are referred to the values in `classes.c`.

Cfengine is easily extensible so as to support a variety of architectures. You can even add your own. To do so you need, first of all, to define a new class for the operating system concerned. The file *classes.c* has been separated off from the remainder of the source code so that you can easily see which data structures need to be extended.

To make life as straightforward as possible, three unused classes have been defined. They are called (unremarkably) *unused1*, *unused2* and *unused3*. If you add any further classes, it will be necessary to increase the constant *clssattr* defined in *cf.defs.h* by one for every new addition. You do not need to change *clssattr* if you simple replace one of the unused classes by a real class.

To see fully the impact of what you need to do, you should make a search for the strings *unused?* in all of the source files. Certain special cases need to be handled for each operating system. For example, the form of the filesystem table is quite radically different on some systems such as AIX. One thing you must do is to fill in the default values for the new operating system in the file *classes.c*.

If you fill in the details for a new operating system before it finds its way into a new release, you might consider sending the details to the bug list in the next paragraph.

8 Example configuration files

Here is a sample from a large configuration file, just to give you some ideas. The file is broken up into manageable pieces for convenience.

8.1 cfagent.conf

```
######################################################################
#
#  CFENGINE CONFIGURATION FOR site = iu.hioslo.no
#
#  This file is for root only.
#
#######################################################################

###
#
# BEGIN cfagent.conf
#
###

import:

   #
   # Split things up to keep things tidy
   #

   any::
                    cf.groups
                    cf.main
                    cf.site
                    cf.motd

   hpux::           cf.hpux
   linux::          cf.linux
   solaris::        cf.solaris
   sun4::           cf.sun4
   ultrix::         cf.ultrix
   freebsd::        cf.freebsd

   #
   # Do you want to do this ?
   #

   AllHomeServers:: cf.users


###
#
# END cfengine.conf
#
###
```

8.2 cf.groups

```
##############################################################
#
# cf.groups - for iu.hioslo.no
#
# This file contains  all group/class definitions
#
#################################################################

###
#
# BEGIN cf.groups
#
###

groups:

   #
   # Define some groups
   #

   iu = ( nexus ferengi regula borg dax lore axis worf daystrom voyager
          aud1 aud2 aud3 aud4 bajor ds9 takpah takpeh nostromo galron
          thistledown rama chaos pc-steinarj pc-hildeh way jart kosh )

   diskless   = ( regula ferengi lore )

   standalone = ( nexus axis dax borg worf daystrom voyager
                  aud1 aud2 aud3 aud4 bajor ds9 takpah takpeh
                  nostromo galron thistledown rama pc-torejo
                  pc-steinarj pc-hildeh )

   AllHomeServers   = ( nexus )
   AllBinaryServers = ( nexus borg )

   XBootServer  = ( nexus )
   WWWServers   = ( nexus )
   FTPserver    = ( nexus )
   NameServers  = ( nexus )
   PasswdServer = ( nexus )
   BackupHost   = ( nexus )

   MailHub      = ( nexus )
   MailClients  = ( iu -nexus )

###
#
# END cf.groups
#
###
```

8.3 cf.main

```
##############################################################
#
# cf.main - for iu.hioslo.no
#
```

```
# This file contains generic config stuff
#
#################################################################

###
#
# BEGIN cf.main
#
###

control:

   access    = ( root )        # Only root should run this

   site      = ( iu )
   domain    = ( iu.hioslo.no )
   sysadm    = ( drift@iu.hioslo.no )

   repository = ( /var/spool/cfengine )

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

   sensiblesize  = ( 1000 )
   sensiblecount = ( 2 )
   editfilesize  = ( 20000 )

   mountpattern = ( /$(site)/$(host) )
   homepattern  = ( u? )

   #
   # If we undefine this with cfengine -N longjob
   # then we switch off all jobs labelled with this class
   #

   addclasses = ( longjob )

   #
   # Macros & constants are inherited downwards in imports
   # but are not passed up to parent files. Good idea to
   # define them all here
   #

   masterfiles = ( /iu/nexus/local/iu )
   main_server = ( nexus )
   cfbin       = ( /iu/nexus/local/gnu/lib/cfengine/bin )
   gnu         = ( /local/gnu )
   ftp         = ( /local/iu/ftp )
   nisslave    = ( dax )
   nisfiles    = ( /iu/nexus/local/iu/etc )

   #
   # The action sequence for daily (full) runs and
   # for hourly updates (called with -DHourly)
   #

   Hr00::
```

```
      actionsequence =
         (
         copy
         mountall
         mountinfo
         checktimezone
         netconfig
         resolve
         unmount
         shellcommands
         addmounts
         links.Prepare
         files.Prepare
         directories
         links.Rest
         mailcheck
         mountall
         required
         tidy
         disable
         editfiles
         files.Rest
         processes
         )

   !Hr00::

      actionsequence =
         (
         resolve
         shellcommands
         copy
         editfiles
         processes
 links
         )

   force::

      actionsequence =
         (
         files.Prepare.Rest
         tidy
         )

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

homeservers:

   iu:: nexus

binservers:

   iu.solaris::                  nexus
   iu.linux::                    borg

mailserver:
```

```
   any:: nexus:/var/mail

mountables:

   any::
         nexus:/iu/nexus/u1
         nexus:/iu/nexus/u2
         nexus:/iu/nexus/u3
         nexus:/iu/nexus/u4
         nexus:/iu/nexus/u5
         nexus:/iu/nexus/u6
         nexus:/iu/nexus/ua
         nexus:/iu/nexus/ud
         nexus:/iu/nexus/local
         nexus:/opt/NeWSprint
         nexus:/opt/AcroRead
         borg:/iu/borg/local
         dax:/iu/dax/local

miscmounts:

   linux||freebsd::   nexus:/iu/nexus/local /iu/nexus/local ro

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

broadcast:

  ones

defaultroute:

   cadeler30-gw

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

resolve:

      128.39.89.10  # nexus
      158.36.85.10  # samson.hioslo.no
      129.241.1.99

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

tidy:

   #
   # Some global tidy-ups
   #

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

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

ignore:                       # Don't check or tidy these directories
```

```
      /local/lib/gnu/emacs/lock/
      /local/tmp
      ftp
      projects
      /local/bin/top
      /local/lib/tex/fonts
      /local/iu/etc
      /local/etc
      /local/iu/httpd/conf
      /usr/tmp/locktelelogic
      /usr/tmp/lockIDE
      RootMailLog

      #
      # Emacs lock files etc
      #

      !*
      /local/lib/xemacs

      #
      # X11 keeps X server data in /tmp/.X11
      # better not delete this!
      #

      .X11

      #
      # Some users like to give a file or two 777 protection here
      # so netsurfers can update a log or counter when running as
      # ‘nobody’
      #

      www

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

disable:

   /etc/hosts.equiv
   /etc/nologin
   /usr/lib/sendmail.fc


###
#
# END cf.main
#
###
```

8.4 cf.site

```
###############################################################
#
# cf.site - for iu.hioslo.no
```

```
#
# This file contains site specific data
#
#################################################################

###
#
# BEGIN cf.site
#
###

links:

   Prepare::

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

   dax::

      /iu/dax/local             +> /iu/nexus/local
      /projects                 -> /iu/dax/local/projects
      /iu/nexus/u1/sowille/data -> /iu/dax/scratch/data

   XBootServer::

      #
      # Set up a /local/tftpboot area where all X terminal
      # stuff will be kept.
      #

      /tftpboot                  -> /local/tftpboot
      /local/tftpboot/td/configs -> /local/tftpboot/td/examples/configs
      /etc/bootptab              -> /tftpboot/bootptab
      /tftpboot/usr/lib/X11/td   -> /tftpboot/td

   NameServers::

      /etc/named.boot -> /local/iu/named/named.boot

   MailHub::

      /etc/mail/sendmail.cf ->! /iu/nexus/local/mail/sendmail.cf

   MailClients.solaris::

      /etc/mail/sendmail.cf ->! /iu/nexus/local/mail/client.cf

   nexus::

/local/bin +> /local/latex/bin

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

disable:

  #
  # We run Berkeley sendmail and the config files are
```

```
  # all under /iu/nexus/local/lib/mail
  #

    /etc/aliases

 WWWServers.Sunday::

   #
   # Disabling these log files weekly prevents them from
   # growing so enormous that they fill the disk!
   #

   /local/iu/httpd/logs/access_log   rotate=empty
   /local/iu/httpd/logs/agent_log    rotate=empty
   /local/iu/httpd/logs/error_log    rotate=empty
   /local/iu/httpd/logs/referer_log  rotate=empty

   #
   # CERT warning, security fix
   #

  any::

    /usr/lib/expreserve

  FTPserver.Sunday.Hr00::

   /local/iu/xferlog rotate=3

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

files:

  Prepare::

      /etc/motd              m=0644 r=0 o=root act=touch
      /.cshrc                m=0644 r=0 o=root act=touch

   PasswdServer::

      /local/iu/etc/passwd m=0644 o=root g=other action=fixplain
      /local/iu/etc/shadow m=0644 o=root g=other action=fixplain

   WWWServers.Rest::

      /local/iu/www                          m=775        g=www act=fixall r=inf
      /local/iu/httpd/conf                   m=664 o=root g=www act=fixall r=inf
      /local/iu/www/cgi-bin-public/count_file m=777 o=root g=www act=fixplain

   FTPserver::

      #
      # Make sure anonymous ftp areas have the correct
      # protection, or logins won't be able to read
      # files - or perhaps a security risk. This is
      # Solaris 2 specific...
      #
```

```
      $(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

   Prepare::

      /etc/shells mode=0644 action=touch

   AllBinaryServers.Rest.longjob::

     /local mode=-0002 r=inf owner=root,bin group=0,1,2,3,4,5,6,7,staff
            links=tidy action=fixall

     /local/iu/RootMailLog  m=0666 action=touch

   dax.Rest::

    /iu/dax/scratch        r=0 o=root mode=1777 action=fixall
    /iu/dax/local/projects r=0 o=root mode=755  action=fixdirs

   nexus::

    /local/mail/sendmail.cf o=root m=444 act=fixplain

    /iu/nexus/ua/robot/.rhosts o=robot m=600 act=touch

    /local/iu/named/pz         o=root  m=644 act=fixall r=1

    /local/latex/lib/tex/texmf/fonts  owner=root
                                      mode=1666
                                      recurse=inf
                                      action=fixall

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

tidy:

      #
      # Make sure the file repository doesn't fill up
      #

      /var/spool/cfengine pattern=*    age=3

      /var                pattern=core age=0  r=inf
      /var/spool/mqueue   pattern=*    age=14 type=mtime

   BackupHost::

      # Here we tidy old backup tar files from the backup area
      # A special tmp area gets cleared every 4 days. The files
      # are created by Audun's backup help script (see shellcommands)

      /iu/nexus/backup1      pat=*  age=7

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

```
shellcommands:

   PasswdServer::

      # Build and install the BSD compatible passwd file
      # from the master passwd/shadow file on Solaris

      "/local/iu/bin/BuildPasswdFiles"
      "/local/iu/bin/BuildGroupFiles"

  BackupHost.Sunday.Hr00|BackupHost.Wednesday.Hr00::

      #
      # Make a system backup of /iu/nexus/u? with Audun's script
      #

      "$(cfbin)/cfbackup -p -f /iu/nexus/backup1 -s /iu/nexus/ud"
      "$(cfbin)/cfbackup -p -f /iu/nexus/backup1 -s /iu/nexus/ua"
      "$(cfbin)/cfbackup -p -f /iu/nexus/backup1 -s /iu/nexus/u1"
      "$(cfbin)/cfbackup -p -f /iu/nexus/backup1 -s /iu/nexus/u2"
      "$(cfbin)/cfbackup -p -f /iu/nexus/backup2 -s /iu/nexus/u3"
      "$(cfbin)/cfbackup -p -f /iu/nexus/backup2 -s /iu/nexus/u4"
      "$(cfbin)/cfbackup -p -f /iu/nexus/backup2 -s /iu/nexus/u5"
      "$(cfbin)/cfbackup -p -f /iu/nexus/backup2 -s /iu/nexus/u6"

  nexus.Sunday.longjob.Hr00::

      #
      # See how much rubbish users have accumulated each Sunday
      #

      "$(cfbin)/noseyparker /iu/nexus/u1 $(sysadm) "
      "$(cfbin)/noseyparker /iu/nexus/u2 $(sysadm) "
      "$(cfbin)/noseyparker /iu/nexus/u3 $(sysadm) "
      "$(cfbin)/noseyparker /iu/nexus/u4 $(sysadm) "
      "$(cfbin)/noseyparker /iu/nexus/u5 $(sysadm) "
      "$(cfbin)/noseyparker /iu/nexus/u6 $(sysadm) "
      "$(cfbin)/noseyparker /iu/nexus/ua $(sysadm) nomail"
      "$(cfbin)/noseyparker /iu/nexus/ud $(sysadm) nomail"

   nexus.longjob.Hr00::

      #
      # Update the GNU find/locate database each night
      #

      "$(gnu)/lib/locate/updatedb"
      "/local/iu/bin/newhomepage.sh"

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

editfiles:

    #
    # cfengine installs itself as a cron job - sneaky! :)
    #
```

```
     { /var/spool/cron/crontabs/root

     AppendIfNoSuchLine "0 * * * * $(cfbin)/cfwrap $(cfbin)/cfhourly"
     }

   FTPserver::

      { /etc/shells

      AppendIfNoSuchLine "/bin/tcsh"
      AppendIfNoSuchLine "/local/gnu/bin/bash"
      }


   XBootServer::

      { /etc/inetd.conf

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

   nexus::

      { /iu/nexus/ua/robot/.rhosts

      AppendIfNoSuchLine "borg"
      AppendIfNoSuchLine "borg.iu.hioslo.no"
      AppendIfNoSuchLine "aud4"
      AppendIfNoSuchLine "aud4.iu.hioslo.no"
      }

   dax::

      { /etc/system

      AppendIfNoSuchLine "set pt_cnt=128"
      }

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

required:

   #
   # Any host must have a /local, /usr/local fs. Check that
   # it exists and looks sensible. (i.e. not empty)
   #

   /$(site)/$(binserver)/local


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

copy:

   #
   # NIS seems broken at IU, so here we use NFS to fudge
```

```
   # a file distribution as a temporary solution. Actually
   # this makes the system work faster without NIS!
   #

      $(nisfiles)/services dest=/etc/services o=root g=other mode=0644
      $(nisfiles)/hosts.deny dest=/etc/hosts.deny o=root mode=0644

   !debian::

      $(nisfiles)/hosts    dest=/etc/hosts o=root g=other mode=0644

   PasswdServer::

      /etc/passwd dest=$(nisfiles)/passwd o=root g=other mode=0644
      /etc/shadow dest=$(nisfiles)/shadow o=root g=other mode=0644

   nexus::

      /local/iu/etc/dfstab dest=/etc/dfs/dfstab  o=root  mode=0744

   solaris.!PasswdServer::

      $(nisfiles)/passwd dest=/etc/passwd o=root g=other mode=0644
      $(nisfiles)/shadow dest=/etc/shadow o=root g=other mode=0600
      $(nisfiles)/group.solaris dest=/etc/group o=root g=other mode=0644

   linux::

      $(nisfiles)/passwd.linux dest=/etc/passwd o=root g=other mode=0644
      $(nisfiles)/group.linux dest=/etc/group o=root g=other mode=0644

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

processes:

      "eggdrop"                           signal=kill
      "irc"                               signal=kill
      "ping"                              signal=kill
      "NetXRay"                           signal=kill
      "netxray"                           signal=kill
      "ypserv"                            signal=kill
      "ypbind"                            signal=kill
      "rarpd"                             signal=kill
      "rpc.boot"                          signal=kill
      "README"                            signal=kill # You don't sh README !

   !XBootServer::

      "bootp"                             signal=kill

   #
   # These processes are not killed every hour, but once a day
   # when cfengine runs at night. Note that there are often
   # hanging pine and elm processes. These programs crash and
   # go berserk, using hundreds of hours of CPU time.
   #

   Hr00::
```

```
      "cron"                signal=hup  # HUP these to update their config
      "inetd"               signal=hup

      "/local/sdt/sdt/bin"  signal=term # For those elektro dudes who forget
                                        # to log out
      "netscape"            signal=kill
      "pine"                signal=kill
      "elm"                 signal=kill


###
#
# END cf.site
#
###
```

8.5 cf.motd

```
##################################################################
#
# cf.motd
#
# This file is used to set the message of the day file on
# each host
#
##################################################################

 #####
 #
 # BEGIN cf.motd
 #
 #####


control:

   #
   # This points to the file containing general text
   #

   masterfile      = ( /iu/nexus/local/iu/etc/motd-master )
   local_message   = ( /etc/motd.local )

editfiles:

      { /etc/motd

      BeginGroupIfFileIsNewer "$(masterfile)"
        EmptyEntireFilePlease
        InsertFile "$(masterfile)"
        InsertFile "$(local_message)"
        PrependIfNoSuchLine "This system is running $(class):$(arch)"
      EndGroup
      }
```

```
#####
#
# BEGIN cf.motd
#
#####
```

8.6 cf.users

Whether or not you perform any special services for users, with or without their consent is entirely a matter of local policy. In a school or college situation, users are often unco-operative and some are even irresponsible. This file shows you what you could do in an environment with inexperienced users, but please don't feel as though you have to be this totalitarian.

```
#################################################################
#
# cf.users - for iu.hioslo.no
#
# This file contains user specific actions
#
#################################################################

###
#
# BEGIN cf.users
#
###

ignore:

    robot

tidy:

   longjob::

     #
     # Some users just don't understand what they are doing
     # and this is safest, allbeit totalitarian
     #

     home                pat=.rhosts                     age=0

     #
     # Tidy up users' home dirs
     #

     home                pat=core            r=inf       age=0
     home                pat=a.out           r=inf       age=1
     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   type=ctime
     home                p=*.log             r=inf       age=2
     home                p=Log.*             r=inf       age=3
```

```
     home                p=CKP               r=inf       age=1
     home                p=BAK               r=inf       age=1
     home                p=log               r=inf       age=0
     home                p=*.o               r=inf       age=0
     home                p=*.aux             r=inf       age=3
     home                p=*.zip             r=inf       age=7
     home/.deleted       p=*                 r=inf       age=0
     home/.wastebacket   p=*                 r=inf       age=14
     home/www            p=*~                r=inf       age=1

     #
     # Clear the big cache files netscape creates
     #

     home/.netscape-cache  p=cache????*         r=inf       age=0
     home/.MCOM-cache      p=cache????*         r=inf       age=0
     home/.netscape/cache  p=*                  r=inf       age=0


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

files:

   AllHomeServers.longjob.rest::

     #
     # Check users files are not writable to the world
     # and there are no stale links (pointing nowhere)
     #

     home mode=o-w recurse=inf action=fixall # links=tidy

     home/.xsession mode=755 action=fixall
     home/.cshrc    mode=755 action=fixall

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

copy:

   Hr00.longjob::

   #
   # Make sure each user has an up to date standard
   # setup.  Cshrc just sources in a big standard file
   # which is kept in ~user/../.setupfiles/cshrc
   # to reduce disk wastage
   #

   $(masterfiles)/lib/Cshrc   dest=home/.cshrc
   $(masterfiles)/lib/tkgrc   dest=home/.tkgrc
   $(masterfiles)/lib/fvwm2rc dest=home/.fvwm2rc


###
#
# END cf.users
#
###
```

8.7 cf.solaris

```
#################################################################
#
# cf.solaris - for iu.hioslo.no
#
# This file contains Solaris specific patches
#
#################################################################

###
#
# BEGIN cf.solaris
#
###

directories:

     #
     # httpd/netscape want this to exist for some bizarre reason
     #

      /usr/lib/X11/nls

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

tidy:

     /var/log  pattern=syslog.* age=0

   MailHub::

     /var/mail pattern=lp       age=0

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

files:

  #
  # If this doesn't exist fork will not work and the
  # system will not even be able to run the /etc/rc
  # scripts at boottime
  #

  /etc/system     o=root g=root m=644 action=touch

  /var/log/syslog o=root        m=666 action=touch

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

copy:

   #
   # Some standard setup files, can't link because
   # machine won't boot if their not on / partition.
   #

   /local/bin/tcsh dest=/bin/tcsh mode=755
```

```
   /local/iu/etc/nsswitch.standalone dest=/etc/nsswitch.conf

  #
  # Our named server uses a newer BIND
  # Put this here so that it will be preserved under
  # Solaris reinstallation
  #

  NameServers::

   /local/iu/sbin/in.named         dest=/usr/sbin/in.named         mode=555
   /local/iu/sbin/in.named.reload  dest=/usr/sbin/in.named.reload  mode=555
   /local/iu/sbin/in.named.restart dest=/usr/sbin/in.named.restart mode=555
   /local/iu/sbin/in.ndc           dest=/usr/sbin/in.ndc           mode=555
   /local/iu/sbin/named-xfer       dest=/usr/sbin/named-xfer       mode=555
   /local/iu/lib/nslookup.help     dest=/usr/lib/nslookup.help     mode=444

  any::
   /local/iu/lib/libresolv.a        dest=/usr/lib/libresolv.a      mode=444
   /local/iu/lib/libresolv.so.2     dest=/usr/lib/libresolv.so.2   mode=444
   /local/bin/nslookup              dest=/usr/sbin/nslookup        mode=444

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

editfiles:

      { /etc/netmasks

      AppendIfNoSuchLine "128.39  255.255.255.0"
      }

      { /etc/defaultrouter

      AppendIfNoSuchLine "128.39.89.1"
      }

      { /usr/openwin/lib/app-defaults/XConsole

      AppendIfNoSuchLine "XConsole.autoRaise: on"
      }

   #
   # CERT security patch for vold vulnerability
   #

   { /etc/rmmount.conf

   HashCommentLinesContaining "action cdrom"
   HashCommentLinesContaining "action floppy"
   }

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

disable:

    /etc/.login  type=file
    /etc/aliases
```

```
   #
   # These files are ENORMOUS, don't let them fill the disk
   #

   Wednesday::

      /var/lp/logs/lpsched rotate=empty

      /var/adm/wtmpx       rotate=empty
      /var/adm/wtmp        rotate=empty

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

files:

    /etc/passwd        m=0644 o=root g=other action=fixplain
    /etc/shadow        m=0600 o=root g=other action=fixplain
    /etc/defaultrouter m=0644 o=root g=other action=touch
    /var/adm/wtmpx     m=0664 o=adm  g=adm   action=touch
    /var/adm/wtmp      m=0644 o=root g=adm   action=touch
    /var/adm/utmp      m=0644 o=root g=adm   action=fixplain
    /var/adm/utmpx     m=0664 o=adm  g=adm   action=fixplain

    /tmp m=1777                              action=fixdirs

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

disable:

   #
   # CERT security patch
   #

   /usr/openwin/bin/kcms_calibrate
   /usr/openwin/bin/kcms_configure
   /usr/bin/admintool

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

shellcommands:

   AllBinaryServers.Saturday.longjob.Hr00::

      #
      # Make sure the man -k / apropos data are up to date
      #

      "/usr/bin/catman  -M /local/man"
      "/usr/bin/catman  -M /local/X11R5/man"
      "/usr/bin/catman  -M /usr/man"
      "/usr/bin/catman  -M /local/gnu/man"
      "/usr/bin/catman  -M /usr/openwin/share/man"
      "/usr/bin/catman  -M /local/X11R5/man"
      "/usr/bin/catman  -M /usr/share/man"

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

```
editfiles:


      #
      # A painless way to add an rc.local script to the rc files
      # under Solaris without having to fight though inittab
      #

      { /etc/rc3.d/S15nfs.server

      AppendIfNoSuchLine "sh /local/iu/etc/rc.local"
      }

      #
      # umask defined when inetd starts is inherited by all subprocesses
      # including ftpd which saves with mode 666 (!) unless we do this
      #

      { /etc/rc2.d/S72inetsvc

      PrependIfNoSuchLine "umask 022"
      }


###
#
# END cf.solaris
#
###
```

8.8 cf.linux

```
#################################################################
#
# cf.linux - for iu.hioslo.no
#
# This file contains debian linux specific patches
#
#################################################################

###
#
# BEGIN cf.linux
#
###

files:

      /etc/printcap m=644 o=root action=fixplain

     #
     # Cert advisories
     #

      /bin/mount         m=755 o=root action=fixall
      /bin/umount        m=755 o=root action=fixall
```

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

  disable:

     #
     # Cert advisories
     #

      /sbin/dip-3.3.7n

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

links:

    /local/bin/tcsh   ->  /bin/tcsh

    /local/lib/mail   ->  /$(site)/$(main_server)/local/lib/mail

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

editfiles:

  #
  # Samba default mode needs to be set...
  #

   { /etc/smb.conf

   ReplaceAll "700" With "644"
   }

  #
  # Linux date is very stupid and needs a very careful
  # TZ definition, otherwise it loses
  #

   { /etc/csh.cshrc

   AppendIfNoSuchLine "setenv TZ 'MET-1MET DST-2,M3.5.0/2,M10.5.0/3'"
   }

  #
  # resolv+ ordering
  #

   { /etc/host.conf

   PrependIfNoSuchLine "order bind"
   }


  #
  # Should have been configured already (!)
  #

   { /etc/ld.so.conf
```

```
     AppendIfNoSuchLine "/usr/X11R6/lib"
     }

   #
   # Kill annoying messages
   #

    { /etc/cron.daily/standard

    HashCommentLinesContaining "security"
    }

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

shellcommands:

  Hr00::

     #
     # Find/locate database
     #

     "/usr/bin/updatedb"

###
#
# END cf.linux
#
###
```

8.9 cf.freebsd / cf.netbsd

FreeBSD, OpenBSD and NetBSD are sufficiently similar to have a single file for all.

```
#################################################################
#
# cf.bsd - for iu.hioslo.no
#
# This file contains bsd specific patches
#
#################################################################

###
#
# BEGIN cf.bsd
#
###

links:

    /usr/spool        ->  /var/spool
    /local/bin/tcsh   ->  /bin/tcsh
    /local/bin/perl   ->  /usr/bin/perl
    /usr/lib/sendmail ->  /usr/sbin/sendmail

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

```
files:

   /usr/tmp mode=1777 owner=root action=fixall

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

editfiles:

   #
   # Comment out all lines to shut up this annoying cfengine-like
   # script, which sends mail every day!!!
   #

   { /etc/crontab

   HashCommentLinesContaining "daily"
   HashCommentLinesContaining "weekly"
   HashCommentLinesContaining "monthly"
   }

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

copy:

      $(masterfiles)/etc/printcap.client      dest=/etc/printcap mode=0644

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

shellcommands:

  Hr00::

    "/usr/libexec/locate.updatedb"
    "/usr/bin/makewhatis /usr/share/man:/usr/X11R6/man"

###
#
# END cf.bsd
#
###
```

8.10 cfservd.conf

```
#########################################################
#
# This is a cfservd config file
#
#########################################################

 #
 # Could import cf.groups here and use a structure like
 # in cfengine.conf, cf.main, cf.groups
 #

control:
```

```
  public = ( /usr/local/publicfiles )

  almost_public = ( /usr/local/almostpublicfiles )

  cfrunCommand = ( /iu/nexus/ud/mark/comp/Tests/cfrun-command )

  MaxConnections = ( 10 )

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

admit:   # or grant:

     $(public) *

     $(almost_public) *.iu.hioslo.no *.gnu.ai.mit.edu

     /etc/passwd *.iu.hioslo.no

     #
     # Who can exec cfengine remotely?
     #

     $(cfrunCommand) *.iu.hioslo.no

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

deny:

     $(public)/special *.moneyworld.com
```

Variable Index

Concept Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

FAQ Index

A

B

C

D

H

I

L

M

P

R

S

T

Table of Contents