cfengine reference

Edition 1.6.3 for version 1.6.3

Mark Burgess
Faculty of Engineering, Oslo College, Norway

Copyright © 1995/96/97/98/99/2000 Mark Burgess

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the section entitled "GNU General Public License" is included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that the section entitled "GNU General Public License" may be included in a translation approved by the author instead of in the original English.

This manual corresponds to CFENGINE Edition 1.6.3 for version 1.6.3 as last updated 1 December 2000.

1 Command reference

In this section you will find each facet of a cfengine program listed together with an appropriate explanation. The commands are presented in alphabetical order for ease of lookup. Use this section in conjunction with the example program See Chapter 5 [Example configuration file], page 123.

1.1 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 1.13 [files], page 50, or a `copy` command, See Section 1.7 [copy], page 27. 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.

1.1.1 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').

1.1.2 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.*

1.1.3 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.

1.1.4 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.

1.2 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
        }
```

1.2.1 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
}
```

1.3 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 1.5.16 [homepattern], page 17. See Section 1.16 [homeservers], page 63.

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)`.

1.4 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 1.5.26 [netmask], page 20.

1.5 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 )
```

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 )

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

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

  myvariable = ( something )
  mymacro    = ( somethingelse )
```

Parentheses are required when making a declaring information in cfengine.

The meaning of each of these lines is described below.

1.5.1 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.

1.5.2 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
   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)
   simplelinks            # check only single links (separate from childlinks)
   childlinks             # check only childlinks (separate from singlelinks)
   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. Here one can also write `singlelinks` which checks only single (not multiply linked) objects, or `childlinks` which checks the remainder (multiply linked) objects. In this way one can separate these two actions if required, though normally this is not necessary.

`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.`*class1*`.`*class2*`.`*class3*

  `"module:mytests.`*class1*`.`*class2*`.`*class3* `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. Note might actually be preferable to define classes returned by modules under `AddInstallables` which is equivalent. 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 1.5.24 [moduledirectory], page 19.

`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 also added to the static routing table. This does not apply to DHCP clients.

`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`'.

`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 1.15 [groups], page 61. 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.

1.5.3 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 1.5.2 [actionsequence], page 10.

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 actionsequence. See next section.

1.5.4 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.

1.5.5 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.

1.5.6 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 1.20 [links], page 67. 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.

1.5.7 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 )
```

1.5.8 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.

1.5.9 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.

1.5.10 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.

1.5.11 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 )
```

1.5.12 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.

1.5.13 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 1.7 [copy], page 27.

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.

1.5.14 ExcludeLinks

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

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

The same facility can be specified for each individual link operation using the `exclude` option See Section 1.20 [links], page 67.

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

class::

```
ExcludeLinks = ( pattern )
```

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

1.5.15 ExpireAfter

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 cfengine will try to kill the cfengine which seems to have hung and attempt to restart the action.

```
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.

1.5.16 HomePattern

```
HomePattern  = ( list of wildcards )
```

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 1.13 [files], page 50, Section 1.28 [tidy], page 85.

`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 1.23 [mountables], page 75. 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 )
```

1.5.17 IfElapsed

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.

1.5.18 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 )
```

1.5.19 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 3.2 [cfrc resource file], page 96.

1.5.20 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 )
```

1.5.21 LockDirectory

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

```
LockDirectory = ( /etc/cfengine )
```

1.5.22 LogDirectory

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

```
LogDirectory = ( /etc/cfengine )
```

1.5.23 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 1.7 [copy], page 27.

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.

1.5.24 moduledirectory

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

This is the directory where cfengine will look for plug-in modules for the actionsequence, See Section 1.5.2 [actionsequence], page 10. Plugin modules may be used to activate classes using special algorithms. See ⟨undefined⟩ [Writing plugin modules], page ⟨undefined⟩.

1.5.25 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 1.13 [files], page 50, Section 1.28 [tidy], page 85.

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 )
```

1.5.26 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 1.5.2 [actionsequence], page 10.

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 1.4 [broadcast], page 8. Cfengine works out the value of the broadcast address using the value of the netmask.

1.5.27 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.

1.5.28 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.

1.5.29 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.

1.5.30 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⟩.

1.5.31 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.

1.5.32 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.

1.5.33 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.

1.5.34 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.

1.5.35 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.

1.5.36 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⟩.

1.5.37 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 in the following cases:

- in the 'to' field of a multiple link action,
- in the 'from' field of a copy action,
- in the directory field of a tidy action,
- in the directory field of the files action,
- in the ignore action.

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

1.5.38 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.

1.5.39 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. See Chapter 4 [Using the help scripts], page 121.

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.

1.5.40 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 )
```

1.5.41 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 1.12 [editfiles], page 39.

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}"
```

1.5.42 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.

1.5.43 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 1.5.18 [Inform], page 18. 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 1.5.18 [Inform], page 18.

1.5.44 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 )
```

1.5.45 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.

1.5.46 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.

1.6 classes

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

1.7 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=silent/fix
                              backup=true/false
                              repository=backup directory
                              stealth=true/on/false/off
                              timestamps=preserve/keep
                              symlink=pattern
                              include=pattern
                              exclude=pattern
                              ignore=pattern
                              filter=filteralias
                              recurse=number/inf/0
                              type=ctime/mtime/checksum/sum/byte/binary
                              linktype=absolute/symbolic/relative/hard/none/copy
                              typecheck=true/on/false/off
                              define=class-list(,:.)
                              elsedefine=class-list(,:.)

                              force=true/on/false/off
                              size=size limits
                              server=server-host
                              secure=true/false
                              purge=true/false
                              syslog=true/on/false/off
                              inform=true/on/false/off
```

`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. If the master and image are directories then all of the child files *which are not directories* are copied from source to destination.

`mode, owner, group`
The file mode, owner and group of the images are specified as in the `files` function See Section 1.13 [files], page 50.

`action` The action may take the values `warn` or `silent`. 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.

`backup` If the `backup` option is set to "false", cfengine will not make a backup copy of the file before copying.

`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 1.5.23 [linkcopies], page 19.

`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.

`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 1.5.14 [excludelinks], page 16.

`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.

`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 `cfd` daemon, and you must make sure that you have defined the variable `domain` in the control section of the '`cfengine.conf`' file. If you don't define a domain you will probably receive an error of the form 'cfengine: Hey! cannot stat file'.

`secure` Has an effect only when used in conjuction with copy from a remote file server. This causes cfengine to use 3DES key encryption and one-time keys on transferred data. This provides good enough privacy for the purposes of system adminstration, but it is not meant to be a super-secure means of encryption. Generally speaking the only case in which this function makes sense is in transferring shadow password files. If you are encrypting the transfer of system binaries, you need your head examining. When encrypt is specified, an md5 checksum is always used to verify the file. Note: the encryption keys required to get files from cfd are those for the user under which cfd is running (normally root). Cfd will not switch keys for other users, so encrypted transfer will not work for all users. If a file *requires* encrypted transfer (secure flag in cfd) then normal users will not be able to collect it.

`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:

number`bytes`
number`kbytes`
number`mbytes`

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`
Switches on/off error messages if source and existing destination files do not match in type, e.g. if a file would overwrite a directory or link.

`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. 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.*

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`.

1.7.1 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.

1.7.2 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.

1.8 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
```

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.

1.9 disks

This is a synonyn for `required`, See Section 1.25 [required], page 81. 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(,:.)
```

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 1.5 [control], page 9.

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 (the default units are kilobytes, but you may specify bytes or megabytes), e.g.

1.10 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 1.13 [files], page 50.

The form of a declaration is:

```
directories:

   classes::

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

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 1.13 [files], page 50. 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.

1.11 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
                        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
```

Cfengine renames a given file by appending the name of the file with the suffix '`.cfdisabled`'. 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 1.13 [files], page 50,

```
files:

   some::

      /var/spool/cron/at.allow =0644 N [root] [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:

number`bytes`
number`kbytes`
number`mbytes`

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.

1.12 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"
      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 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 GNU Regular Expression package. These use GNU regular expressions to search line by line through text and perform various editing functions. 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"

   InsertLine "# cfengine - 2/Jan/95"
   InsertLine "/local/bin/start-xdm"

 EndGroup
}
```

Since the first inserted line matches the predicate on subsequent calls, the grouped lines will only be carried out once.

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`.

`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.

`AppendIfNoLineMatching` *quoted-regex*

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`.

`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, See Section 3.5 [FAQS and Tips], page 99.

`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. New files are created with mode 644, 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 1.23 [mountables], page 75, 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 inform level for this file. Default is on.

`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.

`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.

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

`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.

`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.

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

`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.

`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`,`PercentCommentToLine` `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.

`DeleteLinesAfterThisMatching` *quoted-regex*

`DeleteLinesContaining` *quoted-string*
: Delete all lines containing the exact string quoted.

`DeleteLinesMatching` *quoted-regex*
: Delete all lines matching the quoted regular expression.

`DeleteLinesStarting` *quoted-string*
: Delete all lines beginning with the exact string quoted.

`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 expres-

sion, a warning is only printed in verbose mode, but all edits are immediately abandoned.

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

`ElseDefineClasses`
: See `DefineClasses`

`EndGroup` Terminates a begin-end conditional structure.

`EndLoop` Terminates a loop. See `ForEachLineIn`

`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.

`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 inform level for this file. Default is off.

`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 at the current position in the file. After the insert, the file 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-string*
: Moves the current-position pointer to the start of 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`'.

`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 See Section 3.5 [FAQS and Tips], page 99.

`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`.

`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.

1.13 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.

1.13.1 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

                          define=classlist
                          elsedefine=classlist

                          checksum=md5
                          flags=BSD flags
                          syslog=true/on/false/off
                          inform=true/on/false/off
```

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. 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.

/directory This is the only obligatory part of a file action. This is a directory at which a file search should begin. This may be a single file or a directory. 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 1.13.4 [home directive], page 53.

`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 setuid 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 1.13.2 [Recursion], page 53.

`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 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 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 1.10 [directories], page 34.)

`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`

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. The variable `ChecksumDatabase` should be set in `control:` to the filename of a database file which is used to cache checksum values. Note that it is also possible to use a database file for cfd'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 ⟨undefined⟩ [Checksum Databases], page ⟨undefined⟩.

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.

1.13.2 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).

1.13.3 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.

1.13.4 `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.

1.13.5 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.

1.13.6 Files linkchildren

The `linkchildren` facility is almost identical to that already described under `links`. See Section 1.20.3 [Link Children], page 70. 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
```

1.13.7 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`

1.13.8 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`.

1.14 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(.*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
 }

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

 { 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

`reg|link|dir|socket|fifo|door|char|block`

`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:

`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)"
  }
```

1.14.1 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
```

1.15 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:

   science = ( saga tor odin )

   packages = ( saga )

   AllHomeServers   = ( saga )
   AllBinaryServers = ( saga )

   OIH_servers = ( saga )
   OIH_clients = ( tor odin )

   notthis = ( !this )
```

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 now 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.

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.

As of version 1.4.0, you may use the synonym `classes` for `groups`.

1.16 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 1.23 [mountables], page 75.

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
   riemann:/mysite/riemann/local
   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.*

1.17 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.

1.18 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*.

1.19 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
```

1.20 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 1.13 [files], page 50. 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*

The special variable `$(binserver)` *can be used in* `links`.

1.20.1 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. See Appendix A [Runtime Options], page 151.

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 1.20.5 [Hard links], page 71. 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 1.5.6 [copylinks], page 14. 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 1.7 [copy], page 27.

`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 1.5.14 [excludelinks], page 16.

`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 1.20.2 [Multiple Links], page 69. 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 1.17 [ignore], page 64.

`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 1.3 [binservers], page 6. 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.

1.20.2 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 1.5.30 [repository], page 22.

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. See Appendix A [Runtime Options], page 151.

1.20.3 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 1.13 [files], page 50. Nevertheless, this feature clearly requires some caution and is mainly a spice for advanced users of the cfengine model.

1.20.4 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`'.

1.20.5 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 5 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.

1.21 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 3.2 [cfrc resource file], page 96.

1.22 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
```

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.

1.23 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::

   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`.

1.24 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:

   "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
                          action=signal/do/warn/bymatch
                          include=literal
                          exclude=literal
                          syslog=true/on/false/off
                          inform=true/on/false/off

   SetOptionString "quoted option string"
```

By default, the options sent to ps are "-aux" for BSD systems and "-ef" for system 5. 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 true, 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, command or dot separated list of classes becomes activated if the number of regular expression matches is non zero.

`action=`*signal/do/warn*

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. Further examples may be found in the FAQ section See Section 3.5 [FAQS and Tips], page 99.

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 1.14 [filters], page 56, for more complex searches.

1.25 required

This is a synonym for disks, See Section 1.9 [disks], page 34. 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(,:.)
```

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 1.5 [control], page 9.

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
```

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.

1.26 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`. 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.

1.27 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:

  "command-string"

          timeout=seconds
          useshell=true/false
          umask=octal number
          owner=uid
          group=gid
          define=class-list
          background=false/true
          chdir=directory
          chroot=directory
          preview=true/false
```

The user and group ID's of the process can be set to restrict the permissions of the shellcommand to another user only if cfengine is executed by root. If you set the optional timeout string, then cfengine will abort the shellcommand if it exceeds the specified time-limit in seconds. This can be useful for avoiding situations caused by hung network connections etc.

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 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 timeout applies to each separate iteration, not to the sum of all the iterations.

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, since a root process can break out of such a confinement in principle. The `chdir` option changes the current working directory of the restarted process.

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.

1.28 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=wildcard
                          recurse=number/inf
                          age=days
                          size=number/empty
                          type=ctime/mtime/atime
                          dirlinks=keep/tidy/delete
                          rmdirs=true/false/all/sub
                          links=stop/keep/traverse/tidy

                          define=classlist
                          elsedefine=classlist

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

                          filter=filter alias
                          include=pattern
                          exclude=pattern
```

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*
: 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.

`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 1.13.2 [Recursion], page 53.

`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*

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.

`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.

`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.

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.

1.29 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
```

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.

2 Cfengine script gallery

Here is a gallery of simple-minded scripts to give you ideas for making your own. The absence of explicit testing in cfengine programs also makes these scripts transparent while offering a higher level of checking for no cost to the programmer. Similar shell scripts with this property would be complex indeed.

2.1 User scripts for tidying old files

Here is an example script for tidying old files in your own login area. If you want a long diagnostic, add the option `-v` to the first line of the script, before `-f`.

```
#!/usr/local/bin/cfengine -f
#
# Tidy
#

control:

   actionsequence =
      (
      tidy
      )


tidy:

      $(HOME)         pat=core   r=inf  age=0
      $(HOME)         pat=*~     r=inf  age=1
      $(HOME)         pat=#*     r=inf  age=7
      $(HOME)/code    pat=*.o    r=inf  age=7
      $(HOME)/tex     pat=*.dvi  r=inf  age=7
      $(HOME)/tex     pat=*.log  r=inf  age=7
      $(HOME)/tex     pat=*.aux  r=inf  age=7

      $(HOME)/ftp     pat=*.zip  r=inf  age=7
```

2.2 Controlled opening of files for friends and colleagues

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


control:
```

```
  actionsequence =
     (
     files
     )

  gr = ( myshare )


files:

      $(HOME)       mode=0755 action=fixdirs r=0
      $(HOME)/share mode=0664 action=fixall  r=inf group=$(gr)
```

In this example, first your home directory is opened for the world, then all files in the subdirectory `share` and subdirectories are opened to the group `myshare`. This script could be made to run from a login/logout script of some kind (either `.login` or `.xsession`) so that any new files would automatically be controlled.

2.3 Root script for emergency disk clearing

A straightforward script could be used to clear space in cases where the disk hits the overflow level. This script tidies the whole system, not just the affected disk.

```
#!/local/gnu/bin/cfengine -f
#
# Emergency tidyup!
#
# (Users read their cfengine.rm files to see what got deleted!)
#

control:

site = ( mysite )

mountpattern = ( $(site)/$(host) )
homepattern   = ( u? )

actionsequence =
   (
   tidy
   )


tidy:

      home            pattern=core   R=inf   age=0
      home            pattern=*~     R=inf   age=0
      home            pattern=*.dvi  R=inf   age=1
      home            pattern=*.o    R=inf   age=0
```

```
      /tmp            pattern=*      R=inf   age=0  # could be risky
      /usr/tmp        pattern=*      R=inf   age=0  #      "

ignore:

     .X11
```

2.4 Script for making links

The following script could be used as part of a software installation procedure. Note that the link types can be made relative to the from-link by using `type=relative` See Section 1.20 [links], page 67.

```
#!/tmp/cfengine -v -f
#
# Simple example script to make links
#

control:

  actionsequence = ( links )

links:

 host::

   /usr/local/bin                  +> /usr/local/lib/soft/bin
   /usr/local/X11/lib/app-defaults +> /usr/local/lib/soft/app-defaults
```

It makes links from every binary file in the packages 'bin' directory to the more standard binary directory `/usr/local/bin`. This avoids having to place another search directory into the users' `path` variable. The second statement links the package's application defaults files (for the X-windows system) to a directory in the `XAPPLRESDIR` search path.

This script provides only one way of making the necessary files available to users. It is not the only solution to the problem.

2.5 Ftp server

This script carries out the necessary for setting up a safe anonymous ftp server on a sun workstation running SunOS4.1.

```
#!/local/gnu/bin/cfengine -f
################################################################
#
# Cfengine script to set up an outgoing ftp server under
# SunOS 4.1.*.  Suitable for anonymous access.
#
################################################################
```

```
control:

 addclasses = ( local global )

 actionsequence =
    (
    editfiles.global
    directories
    shellcommands
    files
    editfiles.local
    )

 ftp_root = ( /oih/saga/local/ftp )   # macro for convenience
 ftp_id   = ( 99 )                    # uid/gid for ftp

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

editfiles:

 # Note the file /etc/ftpusers can contain a list of users
 # who can NOT use ftp to access files.

 global::

 { /etc/passwd

 AppendIfNoSuchLine "ftp:*:$(ftp_id):$(ftp_id): (line continues)
Anonymous ftp:$(ftp_root):/usr/ucb/ftp"
 }

 { /etc/group

 AppendIfNoSuchLine "ftp:*:$(ftp_id):"
 }

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

directories:

  $(ftp_root)           mode=0555 owner=ftp
  $(ftp_root)/pub       mode=0555 owner=ftp
  $(ftp_root)/bin       mode=0555 owner=root
  $(ftp_root)/usr       mode=0555 owner=root
  $(ftp_root)/dev       mode=0555 owner=root
  $(ftp_root)/etc       mode=0555 owner=root
  $(ftp_root)/dev       mode=0555 owner=root
  $(ftp_root)/usr/lib   mode=0555 owner=root
```

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

shellcommands:

  "/bin/cp /bin/ls $(ftp_root)/bin/ls"
  "/bin/cp /lib/libc.so.1.8* $(ftp_root)/usr/lib"
  "/bin/cp /usr/lib/ld.so  $(ftp_root)/usr/lib"
  "/bin/cp /usr/lib/libdl.so.1.0 $(ftp_root)/usr/lib/libdl.so.1.0"
  "/usr/etc/mknod $(ftp_root)/dev/zero c 3 12 > /dev/null 2>&1"

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

files:

 $(ftp_root)/bin/ls     mode=111 owner=root action=fixall
 $(ftp_root)/usr/lib    mode=555 owner=root action=fixall r=1
 $(ftp_root)/etc/passwd mode=444 owner=root action=touch
 $(ftp_root)/etc/group  mode=444 owner=root action=touch
 $(ftp_root)/pub        mode=644 owner=root action=fixall

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

editfiles:

 local::

 { $(ftp_root)/etc/passwd

 AppendIfNoSuchLine "ftp:*:$(ftp_id):$(ftp_id): (line continues)
Anonymous ftp:$(ftp_root):/usr/ucb/ftp"
 }

 { $(ftp_root)/etc/group

 AppendIfNoSuchLine "ftp:*:$(ftp_id):"
 }
```

3 Problem solving, bugs, FAQs and tips

3.1 '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 cfengine.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
```

3.2 '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.

3.3 Problems with compilation and installation

Although every effort has been made to make the compilation of cfengine trouble free, you might still encounter some problems where non-standard features are concerned. The differences between systems is still a major headache.

Earlier versions of the GNU/Linux operating system do not have support for some of the facilities which cfengine uses. In particular, the ability to use NIS netgroups is absent from earlier versions. During the installation procedure, the `configure` script tests for this possibility and advises you if the facility cannot be used. You can still use cfengine in this case but netgroups will not be expanded. Another problem with GNU/Linux concerns a special socket call to the TCP/IP network interface. This is a command which configures the static routing table and appears to be absent from all versions of Linux and newer IRIX versions. There are also problems with NetBSD. These features are undocumented and will be fixed as soon as they have been understood! If you are running in verbose mode a warning message is printed, otherwise cfengine will ignore attempts to set a default route on the system.

A number of users have experienced a problem using flex and bison in place of lex and yacc. There appears to be a bug in one of these programs which causes cfengine to compile correctly but misinterpret its configuration files, generating an error of the form

```
cfengine:10:action contains invalid statement
```

for every line! The cure is to collect the latest versions of flex and bison from your nearest GNU site.

On really old systems, the configure program is not able to guess what kind of system you are working on. This is true of SunOS versions 4.0.* and also of BSD 4.3 systems. In such cases, you might be able to compile cfengine by using the autoconf option 'host' to specify the host-type.

```
configure --host=sparc-sun-sunos4.0
```

Some other systems which will compile if forced are:

```
m68k-hp-bsd4.3
?-?-bsd4.3
romp-ibm-aos
?-?-aos
```

On some systems, problems arise when using flex. Flex might generate a lexer file lex.yy.c which defines malloc or some other function to be of a type which conflicts with the system definition. If you obtain such a culture crash, edit the lexer file manually and simply delete the offending definitions, then run make again.

As of version 1.4.0 cfengine tries to link in features based on the Berkeley database library '`libdb`' and the TCP wrappers library '`libwrap`'. If you want to use these facilities, you will have to collect them and install them before compiling cfengine. Some problems have been experienced with the linux version of TCP wrappers. If you experience compilation problems, the best thing to do is to edit '`src/conf.h`' after configuration and remove the line beginning '`#define HAVE_LIBWRAP`'.

Newer solaris systems have ACLs. The ACL features only matured in version 2.5 of solaris however, and there have been some problems with the partial implementation in 2.4. If you obtain error messages about unknown ACL functions, edit the '`config.cache`' file in the cfengine root directory and set the value:

```
ac_cv_header_sys_acl_h=${ac_cv_header_sys_acl_h='no'}
```

If you use the DCE (Distributed computing environment) cfengine will try to compile the ACL extension for DFS. This requires the DCE library to be present on the system on which you are compiling. On some systems it also requires thread libraries to be present. Unfortunately, the autoconf program which generates the Makefiles cannot detect shared libraries, only archive libraries. This means that you need to edit the '`config.cache`' file to compile in this support. Set the following values:

```
ac_cv_lib_dce_main=${ac_cv_lib_dce_main='yes'}
ac_cv_lib_dce_main=${ac_cv_lib_thread_main='yes'}
```

Finally, although the autoconfiguration program appends the same libraries to each executable, the following libraries are required only by the following programs.

```
cfengine   -ldce -lthread -lm

cfd        -ldb -lpthread
```

3.4 Bug reports and suggestions

If you experience a problem with cfengine, find a bug or have another suggestion which you wish to air, you can send your thoughts to the special mail address `bug-cfengine@gnu.org`.

Always think a bit before sending a message to the list. This helps to keep down the traffic improves the signal to noise ratio of your thoughts! Try to solve the problem yourself first and look particularly to see whether your system is clean or whether you have installed software or patches which might conflict with cfengine (I can't really imagine how this would happen—but it might). Always be clear about what type of operating system you are running and whether or not it is a complete installation.

Some vendors have begun the practice of distributing systems without key programs like the C compiler, lex and yacc. If you have this problem, you can pick up GNU replacements gcc, flex and bison from any GNU site.

3.5 FAQs and Tips

Here is a problem solver: an encyclopaedia of suggestions and uses for cfengine as accumulated over the years. If you have a contribution to make, please send it to `help-cfengine@gnu.org`. Format your submission like this:

Q: *How do I do....*

A: Very well thank-you....

The table below is updated as the tips occur to me, or as others contribute their own. Please note that any focusing on particular operating systems is purely a matter of personal usage/experience and should not be interpreted as a reflection of how many 'bugs' these systems may or may not contain.

3.5.1 General

Q: How can I check to see what cfengine will do without going through the whole program using '`-n`'?

A: Run cfengine with options:

```
cfengine -p -d3
```

This just parses the file and dumps the contents of the parser to the output.

Q: *Why doesn't cfengine have classes for each hour, instead of just for days?*

A: It does from version 1.3.20 and upward. The hours are denoted in 24 hour clock notation by `Hr00`—`Hr23`. Other time classes are also possible See ⟨undefined⟩ [Using cfengine as a front-end for cron], page ⟨undefined⟩.

Q: *How can I replace the stupid version of sendmail my vendor ships with my OS with, say, Berkeley sendmail?*

A: First of all, compile your new sendmail in a filesystem which is held separate from the OS, for example '`/local/mail`'. You can keep all the files under this new file tree. Now you need to replace '`/usr/lib/sendmail`' with the new version and '`/etc/sendmail.cf`' or '`/etc/mail/sendmail`' with the new files, so that the system can find them.

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

Q: *How can I prevent big log-files like '*`/var/adm/wtmpx`*' and '*`httpd/access_log`*' from filling up my partitions?*

A: Add a line to disable the files once a week. That way you still get a chance to look at them, but you keep the size down:

```
disable::

   Sunday::

      #
      # Do this to throw away old entries
      #

      /var/adm/wtmpx rotate=truncate

      #
      # Or this to keep the last lot
      #

      /var/adm/wtmpx rotate=1
```

An alternative to using `disable` would be to use `tidy`, but then you lose the file once and for all. Note though, that '`wtmpx`' gets updated all the time, so an age `age=0` is necessary to have any effect at all. Some daemons, like '`httpd`', lose their ability to write to a log file if you rename and create a new file. The `rotate` feature in cfengine preserves the open file handle, fixing this problem.

Q: *How can I fix exports in cfengine?*

A: This is a complicated matter. There are lots of ways to do it. The key is either to edit the file '`/etc/exports`' ('`/etc/dfs/dfstab`' in solaris), or to execute an export (share) command directly from `shellcommands`. Under Solaris 2 this is quite easy owing to the fact that the file '`dfstab`' is just a script itself, rather than a configuration file like the old '`/etc/exports`' file. Since editing is limited and you need to specify a list of hosts which might change in time, one of the following is probably the best bet:

```
shellcommands:

   solaris::

      "/usr/sbin/share -F nfs -o rw=netgroup /var/mail"
```

On non-solaris systems:

```
editfiles:

   { /etc/exports

   AppendIfNoSuchLine "/site/host/fs -access=netgroup"
   }
```

Q: *How can I distribute key setup files to users and keep them up to date?*

A: The copy facility will distribute to all users if you use the `home` directive. For instance, to copy a basic '`.cshrc`' file or '`.xsession`', you could write

```
copy:

   /local/masterfiles/.cshrc     dest=home/.cshrc
   /local/masterfiles/.xsession  dest=home/.xsession
```

Q: *Some users set up their own IRC listen services called "eggdrop" which fill up the disk with all kinds of garbage. How can I kill all these processes?*

A:

```
processes:

  #
  # Most users
  #

  "eggdrop"  signal=kill

  #
  # One wise-guy has renamed the daemon!
  #

  ".*wiseguy.*myegg.*"  signal=kill
```

Q: *My license server keeps crashing! How can I check that it's ok?*

A:

```
processes:

  #
  # BSD - often need long descriptive lines
  #       to find this daemon
  #
```

```
  SetOptionString "-ax"

  # Exactly one should be running

  "lmgrd" matches=1
```

Q: *I want to use cfengine to keep DNS tables up to date, using editfiles. How can I make cfengine automatically restart the name server after the edits?*

A: This can be done in two ways. Probably, you need to update a serial number as well as restarting the daemon. You might use a Makefile to simplify this.

```
control:

  actionsequence = ( editfiles control )

  solaris::
           named = ( /usr/sbin/in.named)
  linux:
  freebsd:
           named = ( /usr/sbin/named )
  sun4:
           named = ( /usr/etc/named )

editfiles:

 # edit files here

shellcommands:

   #
   # If you use make to sort out the details
   #

  "/local/gnu/bin/make -f /local/named/Makefile > /dev/null"
```

Or if you need to explicitly restart the name daemon, you could supplement the above with an explicit restart command (this means you lose the cache),

```
processes:

  "named" signal=kill restart "$(named)"
```

Q: *How can I edit all users' login files?*

A: You can use the 'home' pseudo-variable to iterate over all users' homedirectories:

```
editfiles:

    { home/.cshrc

    # Local fixes

    AppendIfNoSuchLine "alias lp  special-print-command"

    # Security

    DeleteLinesMatching "xhost +"
    }
```

Q: How can I kill all processes except for root processes?

A: The following regular expression matches lines which do not contain the string root:

```
processes:

 "\(root\)\{0\}"  signal=term # or kill
```

Q: How can I make cfengine distribute my '`/etc/motd`' file?

A: You will need a master file which contains the text you want to put on your servers. Let us define a variable 'masterfile' which contains this. This master file needs to be available on all hosts on a common NFS filesystem, for instance. (This will change when remote copying is implemented in cfengine.) Now you can do something like the following script. Note that we define a version number for motd which just prevents cfengine from editing the file every single time. You have to change this version number yourself in the config file to force an update. If you don't care about this, just leave out the Begin..End parentheses.

```
control:

masterfile = ( /usr/local/admin/motd-master )

editfiles:

  any::

     { /etc/motd

     BeginGroupIfFileIsNewer "$(masterfile)"
       EmptyEntireFilePlease
       InsertFile "$(masterfile)"
       PrependIfNoSuchLine "This system is running $(class):$(arch)"
       AppendIfNoSuchLine "$(motd_version)"
     EndGroup
```

```
}
```

Note that, if you want special messages added just for, say, linux, then you can single out linux using a special class, or add a special edit after this one.

Note, if you want to keep the first kernel line in this file, you can change this to:

```
editfiles:

   any::

   { /etc/motd

   BeginGroupIfFileIsNewer "$(masterfile)"
     IncrementPointer "1"
     DeleteLinesAfterThisMatching ".*"
     InsertFile "$(masterfile)"
     AppendIfNoSuchLine "$(motd_version)"
   EndGroup
   }
```

bug-cfengine exchange: (Reply courtesy of David Masterson).

I like cfengine a lot and it helps me very much, but I am a little concerned about security. I'm using cfengine to keep some files like /etc/hosts /etc/printcap /etc/mount etc. up to date. So cfengine is started by root in a cron job and reads its cfengine.conf file and all the other information from a filesystem which is common to all the systems. If now somebody manage to alter the cfengine.conf file he can do everything he wants. Wouldn't it be a good idea to make the cfengine.conf file something like a pgp signed messages, so that cfengine can test if this file was created by the right person? Or are there other tips to make it more secure?

I'm not sure, but I think you're over-reacting or you need to be more specific about where you think the holes are in Cfengine's security. If you follow the tips of any standard systems administrator using cfengine or not, there should be few issues concerning security (ie. if security broke, there would be little chance that cfengine could do anything about it anyway).

Ask yourself some of the standard questions with respect to security on UNIX:

- Who owns your script files?
- Who can update those script files?
- If those script files are updated by the wrong person, has your security been purposely broken or does it just have a hole in it?

If you're still worried about the security of your script (be it a cfengine script or not), you could always adjust your cron script to "decrypt" the script file before executing it (see crypt(1)).

Personally, I think if you've set the permissions on your script files properly, then, if someone breaks into those scripts, they've already broken into your system to a point where they could do what they wanted anyway.

Q: How can I distribute password files in cfengine, but keep certain passwords different on some machines, like I can with NIS?

A: If you keep a file with special local passwords, you can override the password file using `editfiles`. First you use `copy` to get the distributed file, then you edit the file like this:

```
editfiles:

  { /etc/passwd

  SplitOn ":"


  ForEachLineIn "/usr/local/etc/passwd.local"

     ReplaceLinesMatchingField "1"

  EndLoop
  }
```

This means, if the first field of each line in the files matches in both files (and both files have the same column format) then replace the line in '`/etc/passwd`' with the line from '`/usr/local/etc/passwd.local`'.

Q: How can I add entries to a list, like in the fiel '`/etc/group`'?

A: Okay, suppose you wanted to make sure that a special user was in the group '`adm`', you would use a construction like this:

```
control:

    person = ( new-user )


editfiles:

 { /etc/group

 BeginGroupIfNoLineMatching "adm.*$(person).*"
   LocateLineMatching "adm.*"
   AppendToLineIfNotContains ",$(person)"
 EndGroup
 }
```

Q: How can I take backups with cfengine?

A: If you have a spare disk partition, you could make a mirror of the most important files. You would use something like this:

```
control:

     excludecopy = ( *.mp3 *.o *.dvi *.ps *.zip *tar* *.lnk
                     core a.out *.au *.wav .* *.exe *.tgz )

copy:

  BackupHost.Hr21::

    /site/host/home dest=/site/host/backup2/u1 r=inf size=<4mb backup=false
```

for each partition you want to back up.

Q: I am using SAMBA and have windows file system data on my unix machine. When I try to make a backup by remote copying files, cfengine goes into a recursive loop when it meets short cuts.

A: Short cuts do not seem to respect the unix file protocols. They look like directories to cfengine and this causes it problems since they do not parse like directories. Add `*.lnk` to the list of files to be excluded during the copy.

Q: Is it possible to force shellcommands to change its working directory? separating the commands with ";" seems to be not possible. shellscripts should not be used for this purpose.

A: (By Rolf Ebert, ebert@pe-muc.de)

I, too, have the need to pass variables to shellcommands and the shellcommands must be executed in a given directory. Here is how most of my shellcommands look like:

```
# generate MMC configs
 '$(shell) "PUBLIC=$(public); export PUBLIC; cd $(public)/mmc/config;
```

`$(shell)` is defined as '/bin/sh -c'. The actual script to be executed is 'blinksrv.x.cfm.in' which is located in '$(public)/mmc/config'. It generates a file in the same directory. As an input parameter the script needs the environment variable 'PUBLIC'.

Q: Using cfengine with the AFS.

We use AFS, therefore our directory structures look like this: '`/afs/btv.ibm.com/system/current/rs_aix43/...`' and so on. I want to build a cf file that can pull what AFS calls the "sysname" into the equation so I can have one cf file that can get data from the proper rs_aix directory depending on what level of AIX, or SUN for that matter, it happens to run on. If I use cfengine's class structure I would have to have a cf file for each AIX/SUN level rather than one that can handle them all. That is why I want exec to work. Can anyone offer an answer to this type of scenario?

A: Courtesy of Jeff Blaine

We reference `@sys` all the time in our cfengine files. It expands to the current machine's AFS sysname on the fly. It's part of AFS, use it!

```
control:
  #
  # ... stuff deleted ...
  #
  rcf_repos = ( /afs/whatever/our_admin_area/config )
  #
  # ... stuff deleted ...
  #

copy:

  # ... stuff deleted ...
  #
  #  SunOS 4 and IRIX automountd startup file to define /net -hosts
  #
  (sun4|irix|irix64)::
      $(rcf_repos)/@sys/etc/auto.master dest=/etc/auto.master mode=444 \
owner=root group=1 type=checksum
  #
  # ... stuff deleted ...
```

3.5.2 AIX

Q: *Hints about AIX?*

A: Send then to `bug-cfengine@gnu.org`.

Q: I get the error

```
ld: 0711-317 ERROR: Undefined symbol: .pthread_sigmask
ld: 0711-317 ERROR: Undefined symbol: .pthread_mutex_init
ld: 0711-317 ERROR: Undefined symbol: .pthread_mutex_lock
ld: 0711-317 ERROR: Undefined symbol: .pthread_mutex_unlock
```

on AIX 4.2

A: Only AIX 4.3 supports POSIX threads fully. You should compile without thread support.

Q: One of our Sysadmins has noted a limitation with line length under AIX. I'm not sure how easy it is to fix but it might be worth noting it somewhere in the cfengine docs. It appears that on the AIX machines the maximum line length we can use for cfengine files is defined by the constant YYLMAX which is set to be 200. On the Suns this constant is set to be the same as BUFSIZ which is currently set to be 1024. This manifested itself by very unusual behavior as cfengine variables began to be overwritten when line lengths in the config file exceeded 200 bytes. Peter can attest to this. Be forewarned "keep line lengths in cfengine less than 200 if you want them to work on AIX machines" Moral of

the story "AIX users beware" Do you think we could just recompile cfengine and use larger buffer sizes all over, I don't know if this constant is all that should be tweaked or if it is somehow tied into the lexx implementation also, since lexx is used to create the parser for the config files.

A: This is a problem with lex and yacc, not with cfengine. The variable BUFSIZ is a system quantity, not related to cfengine's internal variable bufsize. I would recommend getting bison and flex and doing away with the old lex and yacc from the system. Michael Lachowski reports that this is also a problem with HPUX 10's lex/yacc.

3.5.3 BSDI

Q: The following error occurs while compiling:

```
% ./configure
loading cache ./config.cache
checking host system type... i386-pc-bsdi3.1
checking target system type... i386-pc-bsdi3.1
checking build system type... i386-pc-bsdi3.1
expr: syntax error, last argument was ‘’
test: syntax error: Undefined error: 0
expr: syntax error, last argument was ‘’
test: syntax error: Undefined error: 0
expr: syntax error, last argument was ‘’
test: syntax error: Undefined error: 0
checking whether make sets ${MAKE}... (cached) yes
checking for gcc... (cached) gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
...
```

A:

```
  To correct this for BSDI I replaced the following script fragment from
configure:

------------------------------------------------------------------------
#
# Add to the default list of places in LDFLAGS to compensate for
# ... the configure default value of LIBS on some systems
#
for x in /usr/local/gnu/lib /usr/local/gnulib /usr/local/lib /usr/lib /lib
do
  if test -d "$x"; then
    y=‘expr match "$LDFLAGS" ".*-L$x"‘
    if test $y -eq 0; then
      LDFLAGS="$LDFLAGS -L$x"
```

```
    fi
  fi
done

#
# Add to the default list of places in CPPFLAGS to match LDFLAGS above
#
for x in /usr/include /usr/local/gnu/include /usr/local/include
do
  if test -d "$x"; then
    y=`expr match "$CPPFLAGS" ".*-I$x"`
    if test $y -eq 0; then
      CPPFLAGS="$CPPFLAGS -I$x"
    fi
  fi
done
```

With this script fragment, which successfully executed:

```
#
# Add to the default list of places in LDFLAGS to compensate for
# ... the configure default value of LIBS on some systems
#
for x in /usr/local/gnu/lib /usr/local/gnulib /usr/local/lib /usr/lib /lib
do
  if test -d "$x"; then
    case $LDFLAGS in
    .*-L$x)
      LDFLAGS="$LDFLAGS -L$x"
      ;;
    esac
  fi
done

#
# Add to the default list of places in CPPFLAGS to match LDFLAGS above
#
for x in /usr/include /usr/local/gnu/include /usr/local/include
do
  if test -d "$x"; then
    case $CPPFLAGS in
    .*-I$x)
      CPPFLAGS="$CPPFLAGS -I$x"
      ;;
    esac
  fi
```

```
done
-----------------------------------------------------------------------

   I have not completed compiling, installing, testing yet, but presume I
should be fine from here. Thanks for the help.

-Jeff
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Jeff Reed
    Berkeley Software Design, Inc.
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

3.5.4 HPUX

Q: Problems with line length in lex/yacc.

A: See the FAQ for AIX.

Q: *What is the difference between the classes* '`hpux`' *and* '`hpux10`'*?*

A: In version 10 of HPUX, the file structure is reorganized to look more like SVR4. If you have an HPUX-10 system, the appropriate hardclass is `hpux10` rather than `hpux`.

Q: *I set up the new sendmail but the configuration file doesn't work.*

A: There could be a frozen configuration file around. Try:

```
disable:

   hpux::

      /usr/lib/sendmail.fc
```

Q: *Why don't groups work in HPUX?*

A: HPUX uses the file '`/etc/logingroup`' not '`/etc/group`'. Make a link if you need to:

```
links:

  hpux::

     /etc/logingroup -> /etc/group
```

To encourage some cross-fertilization, here's a sanitized sendmail configuration script that I created for HPUX and HPUX10. (From David Masterson, posted to gnu.cfengine.help). David's script is nice since sendmail was the inspiration for cfengine's name.

```
#!/usr/local/bin/cfengine -f
######################################################
#
```

```
# File:  sendmail.conf
#
# Description:   CFEngine script to setup the sendmail.cf.
#
########################################################

control:

access = ( root )

# Postmaster
sysadm = ( myPostmaster )

# NIS domain and group server
site = ( myserver )

# DNS domain
domain = ( myDNSdomain )

# our gateway host
gtwyhost = ( mygateway )

# sendmail.cf can be big
editfilesize = ( 1000000 )

actionsequence =
(
copy
files
editfiles
shellcommands
)

# disable unwanted classes with "--undefine" option
addclasses = ( maildom mailhst )

################
# bindir - location of sendmail
# libdir - location of current mail files
# cfgdir - location of initial mail files
# etcdir - location of hosts.smtp
# own    - who should own result files
# grp    - what group should result files be in
################
    hpux::
bindir = ( /usr/lib )
libdir = ( /usr/lib )
cfgdir = ( /etc/newconfig )
etcdir = ( /etc )
```

```
own = ( root )
grp = ( sys )

    hpux10::
bindir = ( /usr/sbin )
libdir = ( /etc/mail )
cfgdir = ( /usr/newconfig/etc/mail )
etcdir = ( /etc )
own = ( root )
grp = ( sys )

# disable with "--no-copy" option
copy:

$(cfgdir)/sendmail.cf dest=$(libdir)/sendmail.cf type=checksum
mode=0644 owner=$(own) group=$(grp) force=true

# checks for other important files
files:

$(libdir)/aliases mode=444 owner=$(own) group=$(grp) action=touch
$(libdir)/rev-aliases mode=444 owner=$(own) group=$(grp) action=touch
$(etcdir)/hosts.smtp mode=444 owner=$(own) group=$(grp) action=touch

# disable with "--no-edit" option
editfiles:

    any::

# setup general part of sendmail.cf
{ $(libdir)/sendmail.cf

SetCommentStart '#'
SetCommentEnd ''
ResetSearch "1"
UnCommentLinesMatching "#OP.*" # activate Postmaster
ResetSearch "1"
UnCommentLinesMatching "#DY.*"
ResetSearch "1"
LocateLineMatching "DY.*"
ReplaceLineWith "DY$(site).$(domain)" # set site hiding
ResetSearch "1"
UnCommentLinesMatching "#DS.*"
ResetSearch "1"
LocateLineMatching "DS.*"
ReplaceLineWith "DS$(gtwyhost)" # all-knowing SMTP host
# Ruleset 0 setups
ResetSearch "1"
UnCommentLinesMatching "#R.*user@domain to SMTP relay.*"
```

```
ResetSearch "1"
LocateLineMatching "# try to connect to any host for user@domain"
IncrementPointer "1"
CommentNLines "1"
}

# add Postmaster alias
{ $(libdir)/aliases

SetLine "Postmaster: $(sysadm)"
AppendIfNoLineMatching "Postmaster.*"
}

# setup processing of local domain hosts
    maildom::

{ $(libdir)/sendmail.cf

SetCommentStart '#'
SetCommentEnd ''
ResetSearch "1"
LocateLineMatching "DL.*"
ReplaceLineWith "DL$(domain)"
# Ruleset 0 setups
ResetSearch "1"
LocateLineMatching "# connect to hosts in local domain"
IncrementPointer "1"
UnCommentNLines "1"
}

# setup processing via class S
    mailhst::

{ $(libdir)/sendmail.cf

SetCommentStart '#'
SetCommentEnd ''
ResetSearch "1"
UnCommentLinesMatching "#FS.*"
# Ruleset 0 setups
ResetSearch "1"
LocateLineMatching "# connect to hosts in class S"
IncrementPointer "1"
UnCommentNLines "1"
}

# setup of list of hosts for class S
{ $(etcdir)/hosts.smtp
```

```
EmptyEntireFilePlease
Append "localhost1"
Append "localhost2"
}

# disable with "--no-commands" option
shellcommands:

"$(bindir)/sendmail -bk"
"$(bindir)/sendmail -bi"
"$(bindir)/sendmail -bz"
"$(bindir)/sendmail -bd"

#################
# End of File
#################
```

3.5.5 IRIX

Q: *Hints about IRIX?*

A: Send them to `bug-cfengine@gnu.org`.

3.5.6 LINUX

Q: When I try to compile `cfd` I get this error

```
/usr/lib/libwrap.a(options.o): In function `twist_option':
options.o(.text+0x5f7): undefined reference to `deny_severity'
/usr/lib/libwrap.a(options.o): In function `severity_option':
options.o(.text+0x808): undefined reference to `deny_severity'
options.o(.text+0x81c): undefined reference to `deny_severity'
options.o(.text+0x821): undefined reference to `deny_severity'
options.o(.text+0x826): undefined reference to `deny_severity'
options.o(.text+0x82b): undefined reference to `allow_severity'
make[1]: *** [cfd] Error 1
```

A: There seems to be a problem with the distributed version of the TCP wrappers library. Edit the '`src/conf.h`' file and comment out the '`#define HAVE_LIBWRAP 1`' line. This means that you will not be able to use TCP wrappers security however. You might prefer to collect and compile a new version of TCP wrappers.

Q: *Linux insists on rebuilding the message of the day file each time it boots, but that means I keep losing the messages I leave there.*

A: Add the following to your configuration files to comment out the offending lines in the startup scripts:

```
editfiles:
```

```
linux::

 { /etc/rc.d/rc.S

 HashCommentLinesContaining "motd"
 }
```

3.5.7 OSF

Q: Hints about OSF/1?

A: Send them to `bug-cfengine@gnu.org`.

3.5.8 SUNOS (4.1.*)

Q: *How can I delete the '+' sign from the '`/etc/hosts.equiv`' file to improve security?*

A: Use editfiles to delete it:

```
editfiles:

   sun4::

      { /etc/hosts.equiv

      DeleteLinesMatching "+"
      }
```

3.5.9 SOLARIS 2

Q: *I keep getting a 'bad address' error when cfengine tries to reset the netmask and broadcast address.*

A: This is a bug in the sockets library on solaris. It is supposed to be fixed in solaris 2.5.

Q: *How can I add my own file 'rc.local' to the startup bootfiles automatically?*

A: For example, create a file called '`/local/etc/rc.local`' which looks something like this:

```
#
# rc.local
#
PATH=/local/gnu/bin:/bin:/usr/bin:/usr/sbin; export PATH

#!/bin/sh

if [ "`hostname`" = "net-server" ]; then
```

```
   echo Starting WWW server
   /local/httpd_1.4/httpd -d /local/httpd_1.4

   echo Starting GNU finger server
   /local/etc/fingerd

fi

echo Starting ypbind
/usr/lib/netsvc/yp/ypbind

echo Adding a default route and flushing table

route -f add default my-gateway 1

echo Starting xdm

/local/bin/start-xdm
```

Now add an entry to your '`cfengine.conf`' file like this

```
   solaris::

      { /etc/rc3.d/S99rc-local

      AutoCreate
      AppendIfNoSuchLine "exec sh /local/etc/rc.local"
      }
```

Q: *The solaris installation program creates '`/tmp`' without the sticky bit set, so that any user can delete any files in '`/tmp`'. It also means that a race condition can occur in the kernel which can give away root access to any user!*

A:

Add the following line to the configuration immediately!

```
files:

   /tmp mode=1777 action=fixdirs
```

Q: *The ftp program will not allow me to log in to my own account!*

A: The problem is that your shell is not in the system file '`/etc/shells`'. Add a line something like this:

```
editfiles:
```

```
{ /etc/shells

AppendIfNoSuchLine "/local/bin/tcsh"
}
```

Q: `tcsh` *prints an error message on startup and will not read my '*`.cshrc`*' file.*

A: The problem is the central login file distributed with solaris. '`tcsh`' can't understand it. Add a line

```
disable:

    /etc/.login type=file
```

You might want to replace this with a link to your own file.

Q: *Why does solaris fill up the routing table with hundreds of addresses under the loop-back interface? (see netstat -r)*

A: First of all, get the latest patches for solaris, there are bugs in the kernel of solaris 2.4 which makes this worse. Second, make sure you have a file '`/etc/defaultrouter`' with the IP address of your local gateway, if you don't intend to run your system as a router. For instance:

```
files:

  solaris::

     /etc/defaultrouter o=root g=other m=644 act=touch

editfiles:

   solaris::

      { /etc/defaultrouter

      AppendIfNoSuchLine "xxx.xxx.xxx.1"
      }
```

where `xxx.xxx.xxx.1` is the IP address of your gateway.

Q: When trying to boot the system, solaris fails with the error message: *fork: rescource temporarily unavailable/vfork failed.* The system then claims that there is something wrong with one of the file systems.

A: The file '`/etc/system`' has probably been corrupted. If this file does not exist, solaris establish the kernel properly and will not fork any processes. Things usually die early on in the boot process. This causes the side effect that the first fork the system needs to perform (to check the disk file systems) fails and misinterprets the reason for failure of the command. This makes it look as though something is wrong with the disks. Add a line:

```
files:

   /etc/system o=root g=root m=0644 action=touch
```

Q:

I am currently involved with setting up machines with jumpstart. Jumpstart as you may know allows handsfree installation of solaris. One of the things it allows you to do is specify a "finish" script. I am running cfengine from this script to do the bulk of the configuration. During jumpstart, the root of the machine you are installing is actually under "/a". This leads to problems with cfengine with LOCKFILEDIR and LOGFILEDIR at the very least. It would cause problems with all assumptions cfengine makes about system files too. What would be exceedingly nice would be a command line option to redefine where root is assumed to be. I realize this would be pretty hairy with respect to mounting through cfengine, but it would be very useful. For file editing and such a root prefix macro woudlprobably work ok. Let me know what you think.

A:

Define the filenames

```
$(root)/filename
```

and set `$(root)` to "" or "/a" depending on context? That way you could the above without screwing up other things which might be needed. You can switch off the locks with `-K`. And you could override the '`vfstab`' location for solaris in the resource file.

3.5.10 FreeBSD

Q: *How can I stop my FreeBSD system from running the '`/etc/daily`' script which mails me every single day, week and month?*

A: Add an editfiles command

```
freebsd::

  { /etc/crontab

  HashCommentLinesContaining "daily"
  HashCommentLinesContaining "weekly"
  HashCommentLinesContaining "monthly"
  }
```

Q: Why don't filesystems get mounted in the freebsd version of cfengine?

A: Cfengine fixes the '`/etc/fstab`' file, but has to choose between one of two courses of action when mounting, owing to a bug in the mount command on

FreeBSD machines. Cfengine mounts filesystems each time it runs. On all other supported systems this causes no problems,: once a filesystem is mounted it will not be mounted again. Under FreeBSD however, a filesystem gets mounted again each time mount is run, leading to multiple mount information in the mount table. This causes cfengine to warn the filesystem is mounted many times, and could eventually result in a problem for the FreeBSD machine. The policy is therefore to use mount options which do not cause this behaviour, but an unfortunate side-effect is that newly defined filesystems do not get mounted. You can override the mount options if you want to force multiple mounting.

4 Using the help scripts

The following Perl scripts are included as examples and helpful tools in your system administration package. If you do not have Perl, you should get it — it is a very useful language for system administration.

4.1 cfwrap

It is useful to run cfengine on a daily basis from a `cron` script. Use a line like the following one to start cfengine each night. (Note the curiosities of older BSD cronfiles).

```
0 0 * * * /usr/local/lib/cfengine-3.0/bin/cfwrap cfdaily
```

where *cfdaily* is a script which looks something like

```
#!/bin/sh

CFINPUTS=/usr/lib/cfengine/inputs

/usr/local/bin/cfengine
```

You will need to include full path names to the scripts in the cron file. The syntax for using `cfwrap` is as follows.

```
host% cfwrap mycommand

host% cfwrap cfengine

host% cfwrap script_which_sets_CFINPUTS_and_calls_cfengine
```

When you run cfengine it normally only generates output if something is wrong which needs your attention. If you are running cfengine as a `cron` job then the results of each job are normally mailed back to you — or to root. But this causes problems in a networked environment, since mail to root is usually redirected to some central place which local system administrators cannot access. Moreover, you have no way of knowing which host sent the information. The solution is to use a script as a wrapper. The script simply executes some command and collects the output from that command into a file which then gets mailed to some address.

The address to be mailed to is obtained directly from cfengine by calling it with the *-a* switch. The name of the host running cfengine is prepended to the file before it is sent making it easy to see where each message originated. This is also transferred to the subject header of the mail message. `cfwrap` calls `cfmail` in order to mail the result of the command back to the system administrator.

4.2 cfmail

Because there are no standard mail-agents except for *sendmail*, the wrapper script *cfwrap* calls its own simple mail agent *cfmail* to send the message. Note that the *flags* variable in the script *cfmail* arranges for the mail message to be sent with a return address other

than “root”. This means that if the recipient of the mail should decide to hit ‘r’ for ‘reply’ to reply to the message, you have a chance of getting to see the message before it vanishes along with the rest of the mail to *root* into the same black hole that swallows up all those credit cards, house-keys and odd socks that disappear on a daily basis.

You might have to tweak the scripts slightly to tailor them to your own needs. They are used as follows:

```
host% echo test ....  | cfmail -s "Test message" mark
```

5 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.

5.1 cfengine.conf

```
##############################################################################
#
#  CFENGINE CONFIGURATION FOR site = iu.hioslo.no
#
#  This file is for root only.
#
##############################################################################

###
#
# BEGIN cfengine.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
#
###
```

5.2 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
#
###
```

5.3 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
#
###
```

5.4 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
#
###
```

5.5 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
 #
 #####
```

5.6 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 uncooperative 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
#
###
```

5.7 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
#
###
```

5.8 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
#
###
```

5.9 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
#
###
```

5.10 cfd.conf

```
#########################################################
#
# This is a cfd config file
#
# The access control here follows after any tcpd
# control in /etc/hosts.allow and /etc/hosts.deny
#
#########################################################

 #
 # 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
```

Appendix A 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, if they 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 internal 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.

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

'`-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 5 systems to generate a Bus Error or segmentation fault. The same information is available from the cfengine internal 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.

Appendix B Cfengine network protocol specs

Cfengine uses a simple protocol for communicating via a streams-based tcp connection. This section documents the protocol for anyone who might want to create their own clients or server components to interface with cfengine. Several transfers use a standard buffer size of 4096 bytes. The get-file service uses a character based read interface in which the buffer size is not directly relevant; the size of the get-file buffer is dictated by client-side disk blocksizes.

Each new connection to the remote server daemon must begin with a verification or 'login' string whereby the client identifies itself to the server. This information is used to verify the connection by using a reverse DNS lookup and then a double-reverse lookup. This is the basis of hostname authentication.

The various services are listed below:

Stat file

```
AUTH client-name server-name
SYNCH long-time-string STAT filename

reply with OK: <stat-reply-string>
```

Get file

```
AUTH client-name server-name
GET filename

reply with <stream>, break on zero chars received
or BAD: <message>
```

Opendir Recursive parsing of directory trees poses a technical problem. If cfengine keeps a connection open and send buffers on a need-to-know basis, then the number of daemon connections will mount up and overload the server. If cfengine caches the entire directory on the client side, allowing the server connection to be severed, then the caching could easily fill the memory of the client. The policy chosen is to attempt to cache all names client-side, in spite of the possible memory problem. The reason for this choice is that, even on large filesystems (max size of paritions with 32 bit pointers is 4GB), the sum memory used by every filename is only of the order of a few megabytes, and this is within reasonable modern memory limits. It is assumed that, even 64 bit users will not create filesystems which are much large than this.

```
AUTH client-name server-name
OPENDIR dir-name

reply with <stream>, break on zero chars received
```

In the future it might be useful to stat the file automatically here and cache the value client-side.

Exec

```
AUTH client-name server-name
EXEC option-string
```

```
CLASSES stream terminated with --- (CFD_TERMINATE)

reply with <stream>, break on zero chars received
```

Checksum verification

```
MD5 filename 16 byte sequence

reply with CFD_TRUE for no match (copy) or CFD_FALSE for match (no co
```

Reply formatting

The format of reply messages, except for stream data, is

`OK:` *message*

`BAD:` *message*

A return prefix of 'BAD' implies a failure server-side and the client-side wrapper functions return -1 in this case.

The server daemon currently runs single threaded for all requests except GetFile. Since cfd uses heavyweight processes for general applicablity this avoids unnecessary forking and context switching which would download the server. An upper limit on the number of forks which may be performed is set in the config file. This is mananged using the SIGCHLD signal and a pair of arrays in the master processes (This approach is used to avoid shared memory and semaphore usage which is not portable to many older BSD derivative systems).

Variable Index

-

-a option ... 121
-D option ... 13
-l ... 52, 86
-L ... 67
-N option ... 13
-x option ... 96

.

.cfengine.rm ... 87

/

/etc/host.conf ... 96

+

+ ... 61

A

a= ... 50, 85
action ... 50
actionsequence ... 9, 10
AddClasses ... 13
AddInstallable ... 13
addmounts ... 10
age ... 85

B

backup= ... 29
binserver ... 69, 70
binservers ... 6
broadcast ... 8

C

cf.preconf ... 95
CFALLCLASSES ... 152
cfrc ... 96
checktimezone ... 10
childlinks ... 10
control ... 9
create ... 55

D

directories ... 10
disable ... 10, 36
domain ... 9, 15
DryRun ... 15

E

editfiles ... 10
editfilesize ... 9, 15
empty ... 37
exclude= ... 52

F

FileExtensions ... 19
files ... 10, 50
filter ... 56
force= ... 32
freespace= ... 34, 81

G

g= ... 50
group ... 50
groups ... 61

H

home ... 51
homepattern ... 75
HomePattern ... 17
homeservers ... 63

I

import ... 65
include= ... 52
interface configuration ... 66
InterfaceName ... 18

L

l= ... 50
link ... 50
linkchildren ... 54, 70
links ... 10, 67

M

m= 50
mailcheck 10, 73
mailserver 73
miscmounts 74
mode 50
module: 10
moduledirectory 19
mountables 63, 75
mountall 10
mountinfo 10
mountpattern 19

N

netconfig 10
netmask 9, 20
nfstype 9, 21

O

o= 50
ones 8
owner 50

P

p= 85
pattern 85
processes 10
purge= 31

R

r= 50, 85
recurse 50, 85
RepChar 22
required 10
resolve 10, 82
Restricting access 10
rotate= 37

S

SecureInput 22
sensiblecount 9
SensibleCount 22
sensiblesize 9
SensibleSize 22
shellcommands 10
ShowActions 22
singlelinks 10
site 9, 23
split 23
SuspiciousNames 23
sysadm 9, 24

T

tidy 10, 85
timezone 9
touch 54
truncate 37
type= 37, 71

U

unmount 10, 88

W

Wildcards 85

Z

zeroes 8
zeros 8

Concept Index

, 78

-

--dry-run option 84
-a option 121
-D option 13
-l option 52, 86
-L option 67
-x option 96

.

.cfdisabled 36
.cfengine.rm 87
'.cfnew' files 27
'.cfsaved' files 70
'.cshrc', distributing 101
.X11 directory 64
'.xsession', distributing 101

/

'/etc/defaultroute' 117
/etc/host.conf 96
'/etc/hosts.equiv' 36
'/etc/shells' 116
'/tmp' under solaris 116

+

'+' symbol in '/etc/hosts.equiv' 115

A

Absolute links 68
Access control 10
Access control lists 1
ACL key 2
ACLs 1
action sequence 10
Adding defined classes 13
Adding new classes 97
AFS 21
AFS and cfengine 107
allclasses variable 84
Andrew filesystem 21
AppendToLineIfNotContains, example 105
atime tidies 86
AutoCreate 42
automounter 42
awk, editing 39

B

Backup of files in copy 29
Backups, with copy 106
Bad address error in solaris 115
BeginGroupIfFileExists 42
BeginGroupIfFileIsNewer 42
Berkeley database library 98
Berkeley sendmail 99
Binary servers and links 69, 70
Binary servers, defining 6
Binary servers, priority 7, 69
Bootstrap file 95
Broadcast address 8
Broadcast with solaris 2.4 115
Bugs, reporting 98

C

CatchAbort 43
cd in shellcommands 106
'cf.groups' 124
'cf.main' 125
'cf.motd' 137
cf.preconf bootstrap file 95
'cf.site' 129, 148
CFALLCLASSES 84
Cfengine security worries 104
'cfengine.conf' 123
cfrc resource file 96
cfwrap, wrapper script 121
Class decided by shell command 62
Class dependencies 62
Class information, passing to scripts 84
classes 26, 61, 62
Classes, adding and defining 13
control section 9
Controlling log-files 100
Controlling the size of log files 37
copy 27, 101
Copy, exact filetree images 31
Copying files 27
Creating files 55
cron script to start cfengine 121
ctime tidies 86

D

db library 98
DCE key 3
Deadlock 95
Deadlock zombie bug in restart 78
Declaring classes 13
Decrementing line pointer in editfiles 44
Default route, cannot set 97
'defaultroute' 117
Defining a binary server 6
Defining a home server 63
Defining a mail server 73
Defining a mountable 75
Defining before use 13
Defining classes 13, 26
Defining groups 61
Deleting directories 87
Deleting files 85
Deleting stale links 52, 86
Dependencies 62
Device boundaries 29
Device boundaries and files 53
DFS 21
DHCP 12
Directories, deleting 87
Directories, hidden 19
Directories, making 34
Directory for cfengine modules 19
Directory permissions 53
disable, problems with logging afterwards 100
disable, trimming log files 100
Disabling file types 37
Disabling files 36
disks actions 34
Distributing files 101
Distributing user files 101
DNS 82
domain 15
Domain name 20
Double quotes 40
DryRun 15
Dual homed hosts 66

E

Editing users login files 102
Empty files 86
Environment variable CFALLCLASSES 152
Example configuration files 123
Excluding classes 13
Excluding files from a file sweep 52
exports, fixing 100

F

FAQs 99
Field separator in editfiles 47
File images (copy) 27
File management 50
File sizes, specifying 86
File tree images 31
Files, breaking up into several 65
Files, checking permissions 50
Files, home wildcard 53
Files, importing 65
Files, ownership 54
Files, recursion 53
Files, setting owner 50
Files, syntax 50
Flex and bison problem 97
Force copying 32
Fork error in solaris 117
FreeBSD mount problem 118
freespace= 34, 81
Frequently asked questions 99
Frozen configuration files 110
ftp and alternative shells 116
ftp login problems 116
Full disk warnings 34, 81

G

Group dependencies 62
Group field, editing 47
groups 26
groups 61
Groups, defining 61

H

Hardlinks 71
Help scripts 121
Hints and Tips 99
home directive 53
Home path 17
Home servers, defining 63
home wildcard 51
Homepattern variable. 75
'hosts.equiv' 115
Hour classes 99
How can I make cfengine distribute my '/etc/motd' file? 103
httpd problem with logging 100
Hung machine 95

I

ignore command ... 64
Import files, variables in ... 65
Importing files ... 65
Incrementing line pointer in editfiles ... 44
Interface name, redefining by class ... 18
InterfaceName ... 18
Internal classes, switching off ... 151
Internet address ... 20
IP address ... 20
Iterating over lists in shellcommands ... 84
Iteration over lists ... 23

K

Key, ACL ... 3
Kill processes not owned by root ... 103
Killing processes ... 101
Kilobyte, filesize unit ... 86

L

lex and yacc problems ... 108
Linkchildren ... 54, 70
Links and binary servers ... 69, 70
Links, absolute ... 68
Links, deleting stale ... 52, 86
Links, forcing for non-existent files ... 68
Links, making ... 67
Links, multiple ... 69
Links, removing dead ... 68
Links, single ... 67
Links, traversing in searches ... 52, 86
Linux, installing ... 97
Local disk space, make use of ... 70
Local startup file for solaris ... 115
Log files, controlling the size of ... 37
Log-files, preventing overflow ... 100
Login files, editing for all users ... 102

M

Mail agent ... 121
Mail from scripts ... 121
Mail server, defining ... 73
Making backups ... 106
Making directories ... 34
Making links ... 67
Making paths ... 34
Making use of local disk space ... 70
Master files, updating from ... 101
Megabytes, filesize unit ... 86
Message of the day files ... 103
Miscellaneous mount operations ... 74
Module directory ... 19
Modules, user defined ... 11
Mount paths ... 19
Mountable resources, defining ... 63, 75
Mounted filesystems ... 29
Mounting filesystems ... 74
mtime tidies ... 86
Multiple links ... 69

N

Negating classes ... 12
Negating entries from netgroups ... 61
Netgroups ... 61
Netgroups and Linux ... 97
Netgroups, negating entries ... 61
netmask ... 20
Netmask ... 20
Netmask with solaris 2.4 ... 115
Network interfaces, several ... 66
New systems, support for ... 97
nfs ... 21
NFS mount model and automounter ... 42
nfstype ... 21
NIS, netgroup support ... 61
NT, ACL ... 4

O

ones ... 8
Ownership of files ... 54

P

Password file, editing ... 47
Path to home directories ... 17
Path to mounted filesystems ... 19
Paths, making ... 34
Pattern matching in file sweeps ... 52
Permissions, directories ... 53
Preserving file times in copy ... 29
Previewing shellcommands ... 84
Processes, 0 to 3 ... 77
Processes, check if running ... 101
Processes, checking existence of ... 79
Processes, counting ... 78
Processes, killing ... 101
Processes, signalling ... 79

Q

Quoted strings 40

R

'rc.local' in solaris 115
Recursion in files 53
Relative links 68
Removing directories 87
Removing entries from netgroups 61
Renaming files 36
Replacing file by link 37
Reporting bugs 98
Repository filenames, changing 22
resolv.conf 82
Resolver configuration 82
Resource file 96
Restart zombie deadlock bug 78
Restricting the size of files to be edited 15
rmdirs 87
Rotating log files 37
routed 117
Running cfengine, cron script 121

S

Scripts and class information 84
Scripts, examples 89
Scripts, writing 89
Search patterns in files 52
Searching for home directories 17
Searching, advanced 56
Secure input 22
Security under solaris 116
Security with NIS 115
sed, editing 39
Selecting files in searches 56
sendmail 99, 110
Sendmail configuration 110
'sendmail.fc' 110
Sensible file sizes 22
Sensible limits on files in a directory 22
Setting uid on restarted processes 78
Setting working directory in shellcommands .. 106
Several files 65
sharing filesystems 100
Shell command to decide class 62
Single links 67
Single quotes 40
site 23
size field in disable 38
Solaris and tcsh 117
Solaris routing bug 117
Solaris, '/tmp' 116
Solaris, security 116
Specifying file sizes 86
split 23
SplitOn 47
Starting cfengine, cron script 121
Subnet mask 20
Support for new systems 97
suspiciousnames 23
Switching off backup in copy 29
Switching off internal classes 151
Symbolic links, absolute 68
Symbolic links, relative 68
sysadm 24

T

TCP wrappers 98
tcsh and solaris 117
Tidy by ctime, mtime, atime 86
Tidying empty files 86
Tidying files 85
Time classes, hours 99
timeout= in shellcommands 83
Timeouts during iterations 84
Tips using cfengine 99
Too many open files error 32
Touching files 54
Tree copying, exact 31
Truncating log files 37

U

umask 47, 76, 83
Unmounting filesystems 88
Updating files from master source 101

V

Variables in import files 65

W

Warning about full disks 34, 81
Wildcard home 51
Wildcards in homepattern 17
Working directory in shellcommands 106
Wrapper script 121
wtmpx 100

WWW server logs 37

X

xdev (File system boundaries) 53

Y

yacc problems 108

Z

zeros 8

FAQ Index

/

'/etc/system' missing in solaris. 117

A

Action contains invalid statement problem 97
AFS help 107

B

Bad address error in solaris 115
BSDI configuration problems 108

C

cfd fails to compile 114
Cfengine security worries 104
Changing repository name conventions 22
Configure problem, BSDI 108
copy doesn't always copy files 29
ctime copy doesn't always copy files 29

D

Daily mail in FreeBSD 118
Difference between hpux and hpux10 110
Distribute key set up files to users 101

E

Edit all users login files 102
Edit and restart DNS 102

F

Fix exports in cfengine 100
Fixing /tmp permissions in solaris 116
Fork: resource unavailable in solaris 117
FreeBSD daily mail 118
FreeBSD mount doesn't work 118
Frozen configuration files 110
ftp, can't log in 116

G

Groups in hpux 110

H

Hanging commands, timeouts 83
Hey! Cannot stat file error 30
How can I add users to the '/etc/group' file? . . 105
How can I avoid hanging shellcommands? 83
How can I make cfengine distribute my '/etc/motd' file? 103
How can I override passwords locally? 105
How can I set a timeout for a shell command? . . 83
How do I configure sendmail? 110
How do I set a working directory for a shell command? 106
How to create files while editing 42

I

Iterating over lists 23

K

Kill all processes except root 103
Kill user processes 101
Killing "eggdrop" 101

L

Lex and yacc in AIX/HPUX 108
libwrap problem under compilation 114
License server crashes 101
Line length bug in AIX/HPUX 108

M

Message of the day files 103
Message of the day in linux 114
motd in linux 114
Mount filesystems fails in solaris 117

P

Prevent big log-files 100

R

rc.local in solaris. 115
Remote copy problems, can't stat 30
Replace the stupid version of sendmail..? 99
Routing problem in solaris. 117

S

Solaris /tmp sticky bit 116
Sticky bit in solaris /tmp 116

T

tcsh and solaris error 117
Too many open files error 32

W

Why can't I set a default route? 97
Why doesn't cfengine have classes for each hour..? 99

Table of Contents

1 Command reference 1

1.1 acl 1
1.1.1 Access control entries 2
1.1.2 Solaris ACLs 3
1.1.3 DFS ACLs 3
1.1.4 NT ACLs 4
1.2 ACL Example 5
1.2.1 ACL Example 5
1.3 binservers 6
1.4 broadcast 8
1.5 control 9
1.5.1 access 10
1.5.2 actionsequence 10
1.5.3 AddClasses 13
1.5.4 AddInstallable 13
1.5.5 BinaryPaddingChar 14
1.5.6 CopyLinks 14
1.5.7 DefaultCopyType 15
1.5.8 DeleteNonUserMail 15
1.5.9 DeleteNonOwnerMail 15
1.5.10 domain 15
1.5.11 DryRun 15
1.5.12 editfilesize 15
1.5.13 ExcludeCopy 16
1.5.14 ExcludeLinks 16
1.5.15 ExpireAfter 16
1.5.16 HomePattern 17
1.5.17 IfElapsed 17
1.5.18 Inform 18
1.5.19 InterfaceName 18
1.5.20 FileExtensions 18
1.5.21 LockDirectory 19
1.5.22 LogDirectory 19
1.5.23 LinkCopies 19
1.5.24 moduledirectory 19
1.5.25 mountpattern 19
1.5.26 netmask 20
1.5.27 NonAlphaNumFiles 21
1.5.28 nfstype 21
1.5.29 RepChar 21
1.5.30 Repository 22
1.5.31 SecureInput 22
1.5.32 SensibleCount 22

1.5.33 SensibleSize ... 22
1.5.34 ShowActions ... 22
1.5.35 site/faculty ... 23
1.5.36 SplayTime ... 23
1.5.37 Split ... 23
1.5.38 suspiciousnames ... 23
1.5.39 sysadm ... 24
1.5.40 Syslog ... 24
1.5.41 timezone ... 24
1.5.42 TimeOut ... 24
1.5.43 Verbose ... 25
1.5.44 Warnings ... 25
1.5.45 WarnNonUserMail ... 25
1.5.46 WarnNonOwnerMail ... 25
1.6 classes ... 26
1.7 copy ... 27
1.7.1 Hard links in copying ... 32
1.7.2 Too many open files ... 32
1.8 defaultroute ... 33
1.9 disks ... 34
1.10 directories ... 34
1.11 disable ... 36
1.12 editfiles ... 39
1.13 files ... 50
1.13.1 Syntax ... 50
1.13.2 Recursion ... 53
1.13.3 Directory permissions ... 53
1.13.4 `home` directive ... 53
1.13.5 Owner and group wildcards ... 54
1.13.6 Files linkchildren ... 54
1.13.7 touch ... 54
1.13.8 create ... 55
1.14 filters ... 56
1.14.1 Complete filter examples ... 59
1.15 groups/classes ... 61
1.16 homeservers ... 63
1.17 ignore ... 64
1.18 import ... 65
1.19 interfaces ... 66
1.20 links ... 67
1.20.1 Single links ... 67
1.20.2 Multiple Links ... 69
1.20.3 Link Children ... 70
1.20.4 Relative and absolute links ... 71
1.20.5 Hard Links ... 71
1.21 mailserver ... 73
1.22 miscmounts ... 74
1.23 mountables ... 75

1.24 processes ... 76
1.25 required ... 81
1.26 resolve ... 82
1.27 shellcommands ... 83
1.28 tidy ... 85
1.29 unmount ... 88

2 Cfengine script gallery ... 89

2.1 User scripts for tidying old files ... 89
2.2 Controlled opening of files for friends and colleagues ... 89
2.3 Root script for emergency disk clearing ... 90
2.4 Script for making links ... 91
2.5 Ftp server ... 91

3 Problem solving, bugs, FAQs and tips ... 95

3.1 '`cf.preconf`' bootstrap file ... 95
3.2 '`cfrc`' resource file ... 96
3.3 Problems with compilation and installation ... 97
3.4 Bug reports and suggestions ... 98
3.5 FAQs and Tips ... 99
3.5.1 General ... 99
3.5.2 AIX ... 107
3.5.3 BSDI ... 108
3.5.4 HPUX ... 110
3.5.5 IRIX ... 114
3.5.6 LINUX ... 114
3.5.7 OSF ... 115
3.5.8 SUNOS (4.1.*) ... 115
3.5.9 SOLARIS 2 ... 115
3.5.10 FreeBSD ... 118

4 Using the help scripts ... 121

4.1 cfwrap ... 121
4.2 cfmail ... 121

5 Example configuration files ... 123

5.1 cfengine.conf ... 123
5.2 cf.groups ... 124
5.3 cf.main ... 125
5.4 cf.site ... 129
5.5 cf.motd ... 137
5.6 cf.users ... 138
5.7 cf.solaris ... 140
5.8 cf.linux ... 144
5.9 cf.freebsd / cf.netbsd ... 147
5.10 cfd.conf ... 148

Appendix A Runtime Options 151

Appendix B Cfengine network protocol specs 153

Variable Index 155

Concept Index 157

FAQ Index 163