THEORETICAL PHYSICS

OSLO-TP 9-93

UNIVERSITY OF OSLO

Cfengine V2.0 : A network configuration tool

Mark Burgess

Department of Physics
P.O.Box 1048, Blindern
University of Oslo
N-0316 Norway

E-Mail: theory@physics.uio.no

CONFIGURATION ENGINE V2.0
A network unix configuration tool

Mark Burgess

September 7, 1993

Theory Group
Department of Physics
University of Oslo
P.O. BOX 1048, Blindern
0316 OSLO 3, NORWAY

SYNOPSIS

cfengine -[acdhmnpstv] [-f filename]

SUMMARY

A very high level description language for UNIX machine-park configuration, intended to assist the administration of a local area network by defining the setup of all machines centrally from one file. Designed for portability. To be run uid root.

-h Help information. Display version banner and options summary.

-l Normally *cfengine* does not follow symbolic links when recursively parsing directories. This option will force it to do so.

-v Verbose mode. Prints detailed information about actions and state.

-n No action. Only print what has to be done without actually doing it.

-i Do not attempt to configure the local area network interface.

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

-d Enable debugging output.

-p Parse file and then stop. Used for checking the syntax of a program.

-m Do not attempt to mount file systems or edit the filesystem table.

-c Do not check file systems.

-t Do not tidy file systems.

-s Do not execute scripts.

-a Print only the name of the system administrator then quit.

CHANGES IN V2.0

* All known bugs have been fixed, with the exception of the parsing ambiguities associated with spaces mentioned under "bugs".

* A number of error messages have been improved so as to be less confusing/ more helpful.

* More command line switches have been added for convenience.

* The specification of the mailserver no longer assumes the existence of /usr/spool/mail. The actual directory name on the mailhost is now specified and is always mounted on the "official" directory as specified in the particular release of the operating system concerned. It is now up to the system administrator to define appropriate links to other directories. If mail is mounted in a funny place, a warning is generated if you run in verbose mode, using the -v flag.

* Machines which have static routing tables can opt to define a default route which cfengine will then check. If no default route is defined (such as might be the case at startup) then it will be added to the routing table. If the default route does not match the specification a warning is generated.

* Miscellaneous *nfs* mounts can now be defined, which are not related to the home-server/binserver model. The appropriate system tables are controlled and modified if necessary to ensure that they include any filesystems listed under this directive.

Contents

1 Introduction 5

2 Assumptions and Tips 5

3 An overview of the possibilities 6

4 Log Files 6

5 Invocation 7

6 Compiling 7

7 Program Structure 7
- 7.1 Compound classes 9
- 7.2 Generic class " any" 9
- 7.3 Variables with special meaning 10
- 7.4 Anulling entries when debugging 10

8 Actions 10
- 8.1 localinfo 10
- 8.2 groups 11
- 8.3 broadcast 12
- 8.4 defaultroute 12
- 8.5 shells 13
- 8.6 resolve 13
- 8.7 services 14
- 8.8 mailserver 14
- 8.9 homeservers 14
- 8.10 binservers 14
- 8.11 mountables 15
- 8.12 misc_mounts: Miscellaneous mounts 15
- 8.13 makepath 16
- 8.14 links 16
- 8.15 Link Plus 17
- 8.16 Links: linkchildren 18
- 8.17 Disable: renaming dangerous files 19
- 8.18 files: permissions and ownership 20
- 8.19 Home directories 20
- 8.20 files: setuid root 20
- 8.21 files: options 21
- 8.22 touching directories 22
- 8.23 uid, gid = −1 22
- 8.24 files: linkchildren 22
- 8.25 Ignore: omitting filesystems from recursive file checks 22
- 8.26 TidyFiles 23
- 8.27 required: filesystems with special importance 23

8.28 special: exceptional commands 24
8.29 scripts: shell commands 24

9 Runtime Behaviour **25**

10 Portability **27**

11 Known Bugs **28**

A Simple Example program **29**

B Verbose output **39**

C Wrapper **40**

1 Introduction

To use *cfengine* you create a single file which describes the setup of all machines in a machine-park. The interpreter program "*cfengine*" is compiled on all machines. The program file is distributed to all machines and every one executes the same file. The relevant information is extracted from the file by the interpreter and used to configure each machine individually.

The operation of *cfengine* is essentially class orientated[1]. A program or configuration file consists of a specification of a number of classes together with a number of actions which are to be carried out for each class. For instance, we might want to make a link from /usr/spool to /var/spool on all HP machines. The easiest way to do that would be to define the symbolic link for all machines which own the class "hp". Any HP machine running the program would satisfy the requirements and the link would be made.

In general, actions specified in the program file are performed if the machine which is executing the program file is determined to be a member of the class for which an action is specified. A number of basic system variables are required such as the domain and and the subnet mask on the local area network segment. You choose what types of action are carried out by *cfengine* within the scope of the permitted functionality. You have control over what actions will take place and for which files, machines, groups. Only the order in which events take place is fixed by the interpreter. There is a logical ordering to the required sequence of events (see "Runtime Behaviour" later in this manual).

cfengine is intended first and foremost to be run as a batch job, perhaps daily though there is no limit as to how often it can be run. If run in silent mode (the default) then no output is generated if there is nothing wrong. It is therefore natural to pipe any output to some kind of wrapper program which mails it to the system administrator. See appendix C for an example.

2 Assumptions and Tips

cfengine makes few assumptions, but it is useful to adopt some standard conventions and practices when using it.

- It is assumed that the system possesses a working domain name service which is configured by a file called /etc/resolv.conf.

- Although it is *not* an assumption made by *cfengine*, it makes sense to to mount file systems according to the convention

  ```
  /<faculty>/<machine>/filesystem
  ```

  This has serveral advantages, not least of which is for the sake of being systematic. The actual path may be specified in the two variables *mountpath* and *homepat. See the section 'localinfo' for more remarks about this.*

- In a heterogeneous networked environment, there will be many different conventions for placing the mail directory. *Cfengine* stores the operating system's assumed location for mail internally and will attempt to mount mail there, but it is probably a good

[1] It is not really object oriented since, with the exception of lists, objects cannot be defined.

idea to make links on all machines so that the user (who is caught in the middle of the bizarre conventions used by UNIX manufacturers) has some idea of where he/she is. For instance /usr/mail, /usr/spool/mail, /var/mail, /var/spool/mail are all possibilities. Perhaps all or some of these should be linked together.

3 An overview of the possibilities

Here is an overview of some tasks which can easily be administered by *cfengine*.

- * Management of protection and ownership of files on any filesystem.
- * Monitering of basic security issues: hosts.equiv, shells, setuid root programs.
- * Autoconfiguration of the local area network device interface, netmask and broadcast addresses, as in ifconfig.
- * A symbolic link manager for installing and maintaining symbolic links.
- * Shared filesystem (NFS mount) manager for installing home partitions and binary services from a server.
- * Automatic update of /etc/fstab (or equivalent) with predefined filesystems.
- * Class-controlled execution of user scripts.

The more advanced features include:

- Checking and optional fixing of permissions and ownership in file systems, according to lists of allowed users/groups. The lists may include netgroups.
- Linking all of the files in one directory to another directory in such a way that a mirror image of the directory is maintained in new location. For example: the command `/local/bin +> /local/perl/bin` would link all of the children of the directory /local/perl/bin so that they would appear to be in the directory /local/bin. As new files are added to /local/perl/bin, new links will be automatically made.
- Tidying up garbage files such as "core" files which are older than a certain number of days.

4 Log Files

cfengine stores two types of logs. The first of these is a system log which is stored in /usr/spool/*cfengine*.log. This log contains a list of all known setuid root programs and it used to detect the appearence of new programs which may be a security risk. It also stores a separate log for each user on the system (/.*cfengine*.rm) which contains a list of all the files which were deleted by the tidy function during the last pass. The log is stored in the home directory of each user and gets overwritten each time *cfengine* runs to completion.

5 Invocation

cfengine may be invoked in a number of ways. Here are some examples:

```
% cfengine
% cfengine -f myfile
% cfengine -f myfile -v -n
% cfengine -d
```

The first (default) command looks for a file called `cfengine.conf` in the current directory and executes the commands silently. The second command reads the file `myfile` and works silently. The third works in verbose mode and the -n option means that no actions should actually be carried out, only warnings should be printed.

It is advisable to check all programs with the -n option before trusting them to the system, at least until you are familar with the behaviour of *cfengine*.

There is, in addition to the above, a -l mode which forces *cfengine* to follow symbolic link references. The default is not to follow symbolic links.

The complete list of options is listed in the summary at the beginning of this manual, or you can see it by giving the -h option.

6 Compiling

To compile cfengine you will need: cc, yacc and lex on your system. Edit the Makefile giving the correct options for your system. gcc -traditional may be substituted for cc for versions after 2.2.2 (which are bug ridden and ill-advised). bison and flex may be used instead of yacc and lex with a suitable modification of the Makefile. (yy.tab.c is replaced by prog.tab.c) etc. Note that the solaris include files arriving with the gcc distribution contain an error in sys/dirent.h in the definition of d_name, so this may cause problems I haven't seen yet.

It is assumed that a working resolver exists on the system. The package *resolv+* works well for sun workstations.

7 Program Structure

It is easy to write programs for *cfengine*. Look at the example program in Appendix A for more help. A program consists of a number of declarations of the form.

```
action:

class = ( bracketlist... )
class2 = ( list2 ... )
```

The form of the program is free. Use of space is unrestricted, though sometimes the parser will be confused if there is not a space before a closing bracket etc . A *class* is defined to be one of the following:

- The name of an architecture (a hard class) *e.g.* ds, solaris
- The hostname of a particular host *e.g.* boson, gollum, terminator

- The name of a group, as defined in groups (soft class), *e.g.* sysadm, servers
- An action other than special (3 membered classes only) *e.g.* links

Comments in a *cfengine* program are specified by the number symbol # as in shell programming and apply for the rest of the current line. Actions are reserved words from the following list:

```
localinfo
broadcast
groups
shells
resolve
defaultroute
makepath
misc_mounts
files
ignore
tidy
homeservers
binservers
mailserver
required
mountables
links
special
disable
scripts
```

Any hard class is also a reserved word.

```
ds
sun4
hp
aix
solaris
osf
linux
```

Two hard classes are always defined by the language itself, namely the name of the current host and its operating system architecture. These are determined internally by system calls to *uname(2)*. Thus a declaration of the form

```
mymachine = (list)
```

will always be carried out on the machine "mymachine", and

```
solaris = ( list )
```

will always apply to machines which run solaris. In addition the wildcard class "any" will match any machine type and therefore actions marked "any" will be carried out on all machines regardless of architecture or operating system type. [2]

7.1 Compound classes

In certain action contexts compound classes are used. The next section "Actions" discusses the use of these in more detail. These are formed from two class types and bound by a full-stop or "dot" symbol. binservers supports two membered classes of the form

```
class1.class2 = ( item1 item2 .... )
```

where class1 is a defined group and class2 is a hard class. For instance:

```
mygroup.sun4 = ( sun sol star )
```

special actions contain three membered classes

```
class1.class2.class3 = ( item1 item2 item3 ...)
```

where class1 is a group, class2 is a hard class and class3 is the name of an action (without the trailing colon). For example:

```
group.sun4.links = ( ... )
group.any.files = ( ... )
```

The righthand side have the correct syntax to match the type of action being on the lefthand side. This syntax should normally be reserved for *special* exceptions, since most actions can be defined in a nicer way under the appropriate action heading. In this form the action will be carried out by a given host if each of the classes in the compound class are satisfied by the host (i.e. it is the logical AND of the classes.).

7.2 Generic class " any"

The generic wildcard "any" may be used to replace any hard class. Thus instead of assigning actions for the class sun4 only, one might define actions for any architecture by specifying:

```
any = ( list ... )
```

Note that, since it is possible to interchange hostnames, groups and hard classes in definitions, the wildcard "any" also has the effect of meaning "any host", but this meaning is only incidental.

[2]This wildcard replaces only hard classes and is therefore only substituted for something like sun4, ds in compound classes. In practice this never matters.

7.3 Variables with special meaning

Certain listed items may contain references to the following variables

`<faculty>` The faculty as defined in localinfo

`<binserver>` The default server for binary data.

`<host>` The hostname of the machine running the program

These variables are provided in order to encourage the definition of fully generalized path-names. The judicious use of these variables can reduced many definitions to a single one, with some thought.

7.4 Anulling entries when debugging

A useful trick when debugging is to eliminate unwanted actions by changing their class name. Since *cfengine* assumes that any class it does not understand is the name of some host, it will simply ignore entries it does not recognize. For example:

```
myclass = ( ... )
```

can be changed to

```
Xmyclass = ( ... )
```

Since `Xmyclass` no longer matches any defined classes, and is not the name of any host it will simply be ignored.

8 Actions

The actions performed by *cfengine* are defined using the following contructions. (Look at the example in the appendix for more examples.) A detailed description of the what is done at run-time is given in the next section.

8.1 localinfo

Here we define some general information for the subnet in which this file applies. Here is an example

```
localinfo:

    faculty = ( mn )
    domain  = ( uio.no )
    sysadm  = ( drift@fys.uio.no )
    netmask = ( 255.255.254.0 )
    timezone = ( MET )

    mountpath = ( /<faculty>/<host> )

    homepat   = ( u1 u2 u? <faculty>-home-<host>? )
```

The faculty variable is used in the construction of generic filenames. This information is copied directly into the variable `<faculty>` which may be used later in the definition of symbolic links and required file systems. The domain information is used in the resolver configuration. `sysadm` is intended to be the mail address to which mail will be sent. It is not yet implemented however. The netmask and broadcast addresses must be specified in the decimal network byte order form shown. The timezone should be in upper case, provisionally.

`mountpath` is the pathname under which it is assumed that all mounted file systems belonging to the host lie. In this case, if the hostname were "zaphod" it would expand to `/mn/zaphod`.

`homepat` is a list of names for home directories. Usually this will just be something like `u1`, `u2` etc. This name is added to `mountpath` in order to obtain the absolute path to user home directories. Wildcards may be specified as well as generic variables with the exception of `binserver`. The last expample above would expand to `mn-home-zaphod?`, a perverse choice though, of course, possible. The question mark matches a single character.

Users mounting homedirectories like `/home/machine` are advised to group together all file systems under some logical scheme like the one above (/faculty/machine/filesystem) and to make links to `/home/machine` if they really must, rather than setting mountpath to /home/machine since cfengine checks for other mounted filesystems under mountpath. This feature would then be useless.

It is illegal to define more than one element in all the above lists with the exception of *homepat*.

8.2 groups

Groups are soft classes. There is no limit to the number of groups which may be defined. Machines may appear in as many groups as desired. Remember that groups are classes and should therefore be used to group machines according to their function. For example: you might define the following:

```
servers = ( anyon fidibus hope )
suncluster = ( anyon boson fermion semion gluon gluino)
ypslaves = ( anyon fidibus )
```

The machine anyon appears in all three groups, but this is not a problem. This may be used to advantage. It simply means that the machine anyon inherits the soft classes "servers", "suncluster" and "ypslaves", and thus any actions which pertain to those classes will be carried out for host "anyon".

Groups are defined using the list syntax. The list members may be the names of machines or may refer to netgroups. The NIS `+` or `+@` notation is used to signify a netgroup. For example

```
physics = ( +@physics-sun4-hosts roger bambi thumper +NISothers )
usering = ( bilbo mordor gollum )
```

Note that there is no penalty for using netgroups here, regardless of the size of the netgroups. The members of the group are never stored, for it is sufficient to know that the host name is one of the members in the list. When the host is found to belong to the list on the right hand side of a group declaration, it inherits the class on the left hand side which results in the storage of only one item.

The '-' symbol may be used to make exceptions from a list. This is useful if you have defined a netgroup of many machines but a particular rule should not apply to one or two members of the group.

```
    exception = ( +@my-net-group -somemachine -@sub-group )
```

This syntax means that the group 'exception' consists of all the machines in the netgroup *my-net-group* minus the host 'somemachine' and minus all the members contained in the netgroup 'sub-group'. Note that unlike the case of the '+' symbol, the '@' symbol here changes the meaning from a single entry to a netgroup.

Be warned that if an action is specified for a group which is not defined, *cfengine* will assume that the name refers directly to the name of a host, so that misspellt words will simply be ignored since they will never match either a group or a hostname.

See also the sections on `homeservers` and `binservers`. Note: *There is no need to separate machines into different architectures at this stage; this is done later by defining* `binservers`. cfengine *will only match the correct architecture, in architecture dependent actions. The machines in the groups do not even have to belong to the current domain, but may be machines which the current domain will receive services from, perhaps in the form of NFS file systems.*

8.3 broadcast

Every cfengine.conf file must specify the local convention for forming broadcast addresses. There are two choices: either with "ones" or with "zeroes". If zeroes is specified

```
   any = ( zeroes )  # or zeros, if you spell it like that
```

then the broadcast address is formed according to the rule

```
broadcast = IP address & netmask
```

If ones is specified, it is formed by

```
broadcast = IP address | ~netmask
```

The broadcast address is specfied per machine/group/class.

8.4 defaultroute

Dynamical routing is not configurable in *cfengine*, but for machines which have static routing tables, it is useful to check that the default (wildcard INADDR_ANY) is configured so as to point to the nearest router or gateway. The syntax for this is simply:

```
   class = (
           mygateway
           )
```

or

```
   any = (
         129.240.22.1 # address for my gateway
         )
```

This checking is optional. The effects of this command may be seen by looking at the static routing tables using *netstat -r*. This command does the same as the BSD shell command *route(1)*. It is equivalent to

```
route add default mygateway 1
```

There is a problem with the *ioctl(2)* commands which manipulate the routing tables, namely that the default route may be assigned more than once so any number of different addresses. Moreover, once an address has been assigned, it is not easily removed again. *cfengine* therefore checks before adding anything that the present setup is okay. Only if the default route is missing will it be added. It is not possible to fix badly assigned routing since it requires "routing" in the kernel tables – something which I am not willing to do (at least for the time being). The command *route -f* will flush the gateway references from the routing tables if need be and then you can either fix things manually or run *cfengine* one more time to add the default route.

8.5 shells

The file /etc/shells is generated from the lists given here. Note that since the issue of allowed shells is, at least in principle, a security issue, this file gets generated from scratch each time the program runs. It simply copies the list of shells for a given host machine into the file. If no shells are defined the file will be erased! The generic class "any" is probably appropriate here. Example:

```
   any = (
         /bin/csh
         /bin/sh
         /local/bin/tcsh
         )
```

The file is set to mode 0644.

8.6 resolve

The /etc/resolv.conf file specifies the default nameserver and the default domain. These are currently the only two types of information that *cfengine* cares about. Specifying a list of nameservers (by decimal IP address), for instance

```
   any = (
         129.240.22.35
         129.240.22.18
         129.240.2.3
         )
```

results in a file which looks like:

```
  domain uio.no
  nameserver 129.240.22.35
  nameserver 129.240.22.18
  nameserver 129.240.2.3
  ...
```

That is to say, any line starting with "domain..." gets overwritten by the default domain. If any of the addresses in the list exist they are moved to the head of the list according to the ordering specified. Any other lines are left alone. The editing algorithm will delete lines which begin or end in spaces or blank characters since the resolver cannot understand them anyway, but does not complain. The file is set to 0644.

Different zones in the network could easily be programmed to use a different default nameserver by defining appropriate groups for different zones.

8.7 services

This action is not yet implemented, so you can write what you like here. Syntactically, the items should be path names to daemons and processes that should be running on the host. Example:

```
mymachine = (
             gopherd
             rpc.frameusersd
             /fys/mutils/bin/motdd
             )
```

8.8 mailserver

This is used to look for a mail spool area /usr/spool/mail. It is advised to make links to this area on a system 5 machine, especially in an environment with many different unix varieties since there are many dissimilar conventions for mail directory placement. Only one mailserver may be defined per group. For example:

```
most = ( fidibus )
some = ( ulrik )
```

8.9 homeservers

A list of machines which have disk space for users' home directories.

```
group = ( machine1 machine2 machine3 ... )
```

The hosts do not necessarily belong to the users domain. *cfengine* uses this list to find out what file systems a given machine has permission[3] to mount. Private groups can therefore be defined as desired. A file system is deemed mountable if it comes from a machine which is in a list which matches the acceptable classes for the current host. That is, if *cfengine* finds a list which applies to a group I am a member of, I can mount all the filesystems which originate on machines in that list.

8.10 binservers

A list of machines which have disk space containing binary directories. Binary directories include substiutes for /usr/local, any architecture specific information. The form of the declaration is:

[3]Human permission, not Unix permission!

```
group.hardclass = ( machine1 machine2 machine3 ... )
```

so that both the softclass and platform type (hard class) are specified here. For example:

```
group.sun4 = ( mysun )
```

The hosts do not necessarily belong to the users domain. *cfengine* uses this list to find out what file systems (other than homedirs) a given machine has permission to mount. Private groups can therefore be defined as desired. The same rules for mountability apply as for homeservers, except that the host must also match the correct binary type.

8.11 mountables

This specifies a straight list of all the mountable filesystems which exist for mounting by some machine, including the server it is found on. The items do not belong to any particular class. Mountables are a global commodity. Example:

```
mountables:

   (
   anyon:/mn/anyon/u1
   anyon:/mn/anyon/u2
   anyon:/mn/anyon/u3
   anyon:/mn/anyon/local
   anyon:/mn/anyon/fys
   gluino:/mn/gluino/pc
   fidibus:/mn/fidibus/u1
   fidibus:/mn/fidibus/u2
   fidibus:/mn/fidibus/local
   fidibus:/mn/fidibus/fys
   tema:/mn/tema/u1
   )
```

Note that since mountables are parsed in order, you need to place default binservers with highest priority first. If *cfengine* has been instructed to link a particular binary file to some default binary server (for example by some path which includes the variable `<binserver>`) it will pick the first one that matches. If you want to give priority to a particular binary server, make sure it comes eariler in the list than any other binary servers. If there is only one binary server for each mountable then this matter of priority should never be a problem. If however there were two sun4 machines in the list which both exported a file system called `/faculty/<hostname>/local`, then only the first of these would be used.

The mountables list is used in conjunction with the `homserver` and `binserver` lists in order to determine what file systems should be mounted on a host.

8.12 misc_mounts: Miscellaneous mounts

Quite apart from filesystems associated with home directories or binary services, there might be directories which you simply want to mount even though they don't fit into those categories, for instance a central library server, or an information database. The `misc_mounts:` directive may be used for this. For example:

```
misc_mounts:

  any = (
        infohost:/path/info /local/info ro
        )
```

This command would mount the directory /path/info on the host infohost onto the local directory /local/info, read only. If /local/info doesn't exist it is created. The `rw` option clearly specifies read-write mounting.

8.13 makepath

Makepath declarations consists of a number of directories to be created prior the the creation of any symbolic links. An entire path may be created in a single item. It is not necessary to create each parent directory before creating a child. Cfengine will do this automatically.

Read the section about links carefully: Unix exhibits some peculiarities concerning the creation of links which makes the order in which directories are created important.

Directories may also be created using the "touch" option to the "files" action by specifying a directory using the notation "/pathname/." to indicate that the file is a directory and not a plain file. See files.

The makepath list is a straightforward list of absolute pathnames.

```
mymachine = (
            /arfle/barfle/gloop
            )
```

Directories are created with the default permission 0755 with ownership root. If necessary this may be altered later using the "files" action. The special variables `<host>` and `<faculty>` may be used here.

Note: the creation of a path will fail if *cfengine* finds that one of the links in the pathname was a plain file.

8.14 links

Read this section carefully. The order in which links are defined will affect the way in which directories are created. There are peculiarities in the way in which Unix makes symbolic links.

The link manager is a case where the special variables `<binserver>`, `<host>` and `<faculty>` may be used. When a link is defined it is first checked and made if necessary. When referring to symbolic links the format is

```
   class = ( from_link -> to_file)
```

which creates a pointer from the file "from" to the already existing file "to".

If the "from" link lies in a non-existent directory, then the directory will be created. That is, if we want a link from $/a/b/c$ to a file d then the directories a and b will be created, if they do not already exist. Directories are created with the default mode of 0755 and ownership root. An error may occur if part of path to the link is blocked by the presence of a real file *i.e.* not a directory. *NOTE: this feature requires some caution to be exercised. See the note below.*

The "to" path may contain special variables. For example:

```
any = ( /local -> /<faculty>/<binserver>/local )
```

in which case *cfengine* looks for a file system which matches from the list of binservers, starting with the name of the current host. If the link already exists but points somewhere else and warning is issued. If the link exists and is correct, nothing is done.

There is a peculiarity associated with symbolic links. Consider the following shell commands:

```
mkdir /mnt
ln -s /mydir /mnt
```

These commands do not fail with an error but result in a link being made between `/mnt/mnt -> /mydir`. This is a feature/bug which is built into the unix call symlink() and thus it also applies to *cfengine*. This has some implications for link making. *Because* cfengine *makes the directories leading up to the* **from** *link* the following two definitions result in different link structures.

```
mygroup = (
          /local/math -> /mn/gluon/local/math
          /local -> /mn/anyon/local
          )
```

and

```
mygroup = (
          /local -> /mn/anyon/local
          /local/math -> /mn/gluon/local/math
          )
```

In the former case, the directory /local does not exist and is therefore created by *cfengine* in order to be able to make the link. Since /local then exists and is a directory, the second link results in `/local/local->/mn/anyon/local` and not what was actually specified. The second example yields the correct result.

8.15 Link Plus

cfengine enables the definition of multiple links from a single command. Consider the notation:

```
/path +> /varpath
```

This is interpreted to mean: link all the children (files and directories) of /varpath to files with corresponding names in /path. /varpath may contain the special variables `<binserver>`, `<host>` and `<faculty>`.

8.16 Links: linkchildren

The linkchildren feature is an extension of multiple links. The linkchildren construction may be specified in two places: (i) as a "link" action or (ii) as a "files" action. If specified as a link action it takes the form:

```
/my/file/path +> linkchildren
```

This syntax has the following meaning: make links of all of the children in a corresponding file system on a binary server. That is: first look for a binary server in the list of mountables which matches the path name in some way; if such a server is found, link all the files on that server to images in the directory /my/file/path.

For example: we might like to have

```
/local -> /mn/<binserver>/local
```

but actually have a real directory /local with some but not all objects required already filled in. The following

```
/local/priss
/local/lib
```

are real directories, but we want to fill in a full /local filesystem from some server. i.e. we want to link any files that do not exist to mirror images on the server.

Once a suitable server has been found, *cfengine* descends into the directory (one level only) and links any files that don't exist on the host. In order for a server file system to match an item in "mountables", the following conditions must be met.

(i) The potential file system must belong to an allowed binserver

(ii) The directory name (the last element in the path on the host) must exist as an element in the path of the potential item in "mountables".

so that

```
/local
```

on a sun would match

```
/mn/anyon/local      (sun)
```

but not

```
/mn/fidibus/local    (dec)
/mn/anyon/fys        (sun)
```

`linkchildren` does *not* take any notice of home directories in the list of mountables. The process of locating a server works by reverse parsing the name declared in files. For instance:

```
group = (
        /local/lib/emacs +> linkchildren
        )
```

would start a search for a mounted filesystem ending in emacs, then for one ending in lib, then for one ending in local until a match was found. A match is only made for filesystems which reside on valid "binservers" . Binservers are chosen sequentially from the list defined above (not including the current host). Once a valid file system has been identified, *cfengine* checks that the files or required directories exist on the file system and tries to link them. If it turns out that it cannot link the files for some reason (files don't exist) then it continues looking for other alternatives and eventually gives up when there are no more binservers left, generating the message:

```
cfengine: Couldn't link the children of <file> to anything because no
          file system was found to mirror in the defined binservers list.
```

This feature has proven useful on client machines which have their own /local file system, but which have a natural server in addition. Normally one would simply mount the server onto the client. If the client has disk space and is host to some special software, then it is convenient to define a real directory /local on the client, containing the special software and link the remaining files. Example: server anyon has a file system /mn/anyon/local which it exports for all clients. Host gluon has its own filesystem /mn/gluon/local which contains the Mathematica program, because it is the only machine which is lisenced for that package – also it has some spare disk space. We therefore define:

```
gluon = ( /mn/gluon/local +> linkchildren)
```

The file system /mn/gluon/local then contains the package mathematica and links to all of the packages on the server machine anyon. (It is assumed that anyon is correctly defined as a binserver for gluon.)

8.17 Disable: renaming dangerous files

Plain files (not directories) may be disabled by placing them in a list here. The files are moved (renamed) by adding the suffix ".*cfengine*-disabled".

```
group = (
        /etc/hosts.equiv
        )
```

Note that items are disabled before file checking is performed so that files may be emptied by first disabling them and then touching them.

```
disable:

  some = ( /var/spool/cron/at.allow )

files:

  some = ( /var/spool/cron/at.allow =0644 N [root] [wheel] touch )
```

8.18 files: permissions and ownership

The file checking utility is one of the more powerful features of *cfengine*. The idea is to specify a filesystem or simple file which is to be checked for permission, ownership and group ownership. An action is also specified which is then carried out if the file(s) do(es) not fit the specified criterea. This is most easily seen in an example.

```
group = (
        /local/lib -2002 R [root,daemon] [daemon,sys] fixall
        )
```

This means that for machines in "group", we should check the files lying in /local/lib. The R means that the checking should extend recursively into all subdirectories. The first square bracket is a list of the allowed owners of files. This list may either contain textual names or numerical user IDs. The second square bracket represents group ownership, where the same rules apply. The final token is a reserved word which means that *cfengine* should fix the problem without reporting an error.

What happens is the following. The UIDand GID of the file are determined and compared to the owner lists. If there is a mismatch, the UID or GID or both are set to the first name appearing in the list. In addition the file's access modes are checked against the template -2002. The minus sign means that none of the marked flags (specified in the normal chmod octal notation) should exist. *cfengine* unsets them. If we had used a plus + sign, then *cfengine* would ensure that those flags *were* set. An equals sign = means set to the absolute mode. Thus in the example, we are checking that no files are writable to the world and that no files are setuid root. If you want to make sure that some flags are present AND that some are not, then the +- notation can be used. For instance

```
   /fys/drift                    -0002+0770 R [root] [mndrift] fixall
```

Directories: because directories must be executable in order to be readable, cfengine will always add an execute flag to a permission if the corresonding read flag is set. Thus setting a whole file system to mode = 644 *would set all directories to* = 755 *and all plain files to* = 644.

8.19 Home directories

The special directive "home" may be used instead of an absolute file path. If this variable is used, *cfengine* will automatically iterate over all home directories on the current machine. If none are found then no error is generated; it is thus straightforward to make a generic check of all home directories.

```
   any =
      (
      home        -2002 R [*] [mn] fixall
      )
```

8.20 files: setuid root

Files which are setuid root are logged in the system log file /usr/spool/*cfengine*.log and the appearence of any new files results in a warning message. The exception is if the files are marked "fix" then a warning if issued but the name is not stored.

8.21 files: options

The syntax for files is as follows:

```
group = (
        /pathname [+-=]0000 [N|R] '[' UID list ']' '[' GID list ']'
                                                (fix|warn|touch|linkchildren)
        )
```

0000 is a four digit octal number.

A uid/gid list is specified inside square brackets with commas between the members

e.g. `[root,mark,8245,10987]`

e.g. `[*]`

+	add flags
-	remove flags
=	absolute permission

The wildcard `*` may be used instead to match any UID or GID. This is useful for parsing user areas for instance. Note that even though no specific UID or GID is specified, this does not compromise the checking of setuid root files. If "fix" is selected and the first member of the file list is a wildcard the ownership of the file is not changed, but the file is touched. The uidlist may conatin references to netgroups defined in the Network Information Service, using the `+` and `+@` notation.

`[root,+admin]`

Netgroups are not allowed in the gidlist. Allowed file actions are

```
warn
fixall
fixplain
fixdirs
touch
linkchildren
```

If `warn` is specified, only a warning is issued and nothing more is done to files. If `fixall` is specified any problem is immediately corrected regardless of the file type. Plain (regular) files or directories may be singled out by using `fixplain` and `fixdirs` respectively (see also the note below concerning directories). If the target is a file or directory and the action is `touch` then the object will be created and the permission and ownership set to the specified values. If only `+` or `-` are specified then objects are created with the default mode of 755 before adding or removing the named flags. If an object exists its access times will be updated. The special option `linkchildren` is covered below.

Directories: because directories must be executable in order to be readable, cfengine will always add an execute flag to a permission if the corresonding read flag is set. Thus setting a whole file system to mode = 644 *would set all directories to* = 755 *and all plain files to* = 644.

8.22 touching directories

A directory or path name may be specifed by the following line

```
/pathname/. [+-=]xxxx [N|R] [UID] [GID] touch
```

The trailing dot tells *cfengine* that you want to create a directory. All directories up to the final one are created using the standard modes and then the final link in the path is changed to the mode/owner as specified in the UID and GID parts. The recursive function has no meaning here since a newly created directory has no children to parse, but syntactically something must be specified.

8.23 uid, gid $= -1$

When *cfengine* says it is changing the uid or gid of a file to -1 it means that it is not changing it at all. -1 is used as an internal representation of a wildcard. This enables either the uid or gid but not both to be changed easily within the context of the program.

8.24 files: linkchildren

The linkchildren feature is discussed in detail under "links" above. The syntax here is

```
   group = (
           /local/lib/emacs +0000 N [blueeyes] [cataract]  linkchildren
           )
```

The links are made in the usual way, but now the ownership is changed directly to the values given here. In this case the owner of the links would be *blueeyes* and the group would be *cataract*. In most circumstances the ownership root will be appropriate.

Since file processing is performed after all links from the links section have been made (regardless of the ordering in the program file) this option is able to take advantage of those links.

8.25 Ignore: omitting filesystems from recursive file checks

Ignore items are used to explicitly exempt files from checking when recursively parsing directories for files. Files which are specified by direct pathname are not checked against this list of files, thus particular files may be checked even though they reside in a directory which is marked "ignore". The items in this list may be either floating wildcards or pathnames. For example:

```
   any = (
         !*
         /local/lib/gnu/emacs/lock
         )
```

8.26 TidyFiles

The purpose of this action is to purge the system of excess or unwanted files. Rules for deleting files are specified by wildcards and minimum file ages since last accessed (in days).

There are two types of entry here. A program may specify an absolute pathname for files in which case that path and all of its children will be parsed. Additionally it may specify a floating wildcard, that is a wildcard which has no absolute pathname, in which case the tidy rule is applied to all the home directories and their children. Since it is mostly users which generate garbage files, the user home directories are the default paths. If a directory is specified in place of a wildcard, an error results.

Wildcards behave like true UNIX wildcards except when they are floating free of any special path. To give an example, an absolute path

```
/tmp/*  >3
```

will not match `/tmp/.*`, but a floating wildcard

```
*.dvi  >7
```

(which is assumed to belong to the user area `<mountpath>/<homepat>`) will match even files that begin with a dot. This is to allow certain programs such as window managers to hide files in `/tmp/.X11`-like directories. The removal of such files could result in loss of window data about the open display. Also when descending directory trees, directories which begin with "." will not be entered for absolute paths, but *will* be entered for floating wildcards.

The backslash character is ignored if it is the first character of a wildcard. This enables you to write:

```
\#* >1
```

The hash symbol would otherwise be ignored as a comment line.

Note: only the last element of a pathname may be a wildcard.

8.27 required: filesystems with special importance

This check is performed after *cfengine* has mounted all the filesystems it thinks are missing on a given host. Files marked here are deemed to be important for the operation of the host. If they do not exist then a warning is generated. The special variables `<binserver>`, `<host>` and `<faculty>` may be used in these items. The latter two are substituted directly from from variables defined in localinfo and from the information determined from the host. The variable `<binserver>` is expanded according the list of defined binary servers, starting with the host name. This allows a kind of wildcard notation. If the destination does not exist, *cfengine* chooses the next binserver in the list until it finds a match. If no match is found there is an error. For example:

```
required:

   any =    (
          /<faculty>/<binserver>/local
          )

   teorfys = ( /mn/<binserver>/pc )
```

Some checking is also performed on the files here to see if they are sensible. These checks are somewhat arbitrary and are controlled by the constants

```
#define SENSIBLEFILECOUNT 10
#define SENSIBLEFSSIZE    2000
```

If a required directory exists but has fewer than 10 files in it, a warning is issued. Similarly, if a required file exists but is smaller than 2000 bytes, a warning is issued. In a later version one might set these values separately for each item.

8.28 special: exceptional commands

This is a miscellaneous collection of actions specified by three membered classes of the form

```
   group.hardclass.action = ( list ...)
```

Examples:

```
   mygroup.sun4.required =
      (
      /local/lib/sun4libraries
      )

   mygroup.any.links =
      (
      /public -> /<faculty>/<binserver>/public
      )
```

8.29 scripts: shell commands

The scripts or shell commands listed in this section are executed at the end of the program execution. Any shell commands may be given here. The quoted lines are passed directly to the shell (normally */bin/sh*). The syntax is as follows:

```
scripts:

  class = ( list )
```

For example:

```
  any =     (
            "/local/etc/config/use.daily"
            "/fys/mutils/bin/noseyparker /mn/anyon/u1 nomail"
            )

  sun4 = (  "/usr/lib/find/updatedb" )
```

9 Runtime Behaviour

The runtime behaviour *cfengine* is as follows.

- Edit shells file
- Edit resolv.conf file
- Configure network interface
- Mount File systems
- Get info on mounted file systems
- Mount any new file systems which are required
- Makepaths
- Links: extended links and linkchildren
- Links: normal
- Check mail server
- Mount file systems again
- required: check for filesystems
- Tidy files
- Disable
- Files: check permission and ownership, touch and linkchildren.
- Scripts

Note that the linkchildren feature (present both in the link and file actions) is dealt with according to the actions as shown. Linkchildren definitions are not grouped together. This gives the user two shots at linking, before and after other links/directories are made.

A more detailed account of the runtime actions is as follows:

1. Execution begins and *cfengine* obtains name information about the system. Calls to uname(2) provide the name of the machine and release information about the operating system. This information is matched against a number of wildcard templates stored in an internal array and it reduced to a single hard class in the set (sun4, ds, hp, solaris, linux, osf). The system clock is read and the timezone is checked against that defined in localinfo. If there is a mismatch a warning is issued. (In a later version it is intended that the timezone should be fixed automatically.)

2. The parser reads in the user's program and stores the specified action which match defined classes. No actions are carried out during parsing other than to store the information. If any syntax errors occur, execution stops after parsing has finished, or after a maxmium of ten syntax errors. Certain errors generate fatal errors during parsing in order to amplify their significance.

3. Checking is performed to determine whether there is enough information in order to be able to proceed.

4. The files /etc/shells and /etc/resolv.conf (or their equivalents) are checked against the information defined in the shells and resolve headings. Since the shells file is a security issue, in principle, the /etc/shells file is simple over written with a list of the shells which are defined in the user program. The resolver file is edited more carfully. The previous file is moved to *.old. The local domain is set to the value prescribed in the domain variable and it is assured that any entries specified in the program take priority over any existing entries. If the entries specified in the program already exists, they are moved up to their correct place in the file. Entries are not duplicated. No error checking is performed on the entries in resolve, so it is the users responsiblilty to ensure that the address es are well formed internet addresses, written in decimal, network byte order.

5. The network ethernet-interface configuration is read and sanity checks are performed on the IP address of the host. This is a slightly dubious feature since it may not work if the network interface is so badly configured that contact with the network is impossible. The -i option can be used to disable this feature. *cfengine* attempts to retrieve the IP address of the host from a local nameserver and compares it to the address configured in the interface. In principle, *cfengine* could set the IP address, but it is likely that if it is not already set then the machine would be in such a mess that it would not even be able to retrieve information from the network nameserver. Indeed amny interfaces do not permit the flow of packets until the interface has been assigned a proper address. Should this comparison actually manage to retrieve the information and fail, a warning is issued. No attempt is currently made to reset the address. Finally the netmask and broadcast addresses are checked. These are less important and should not cripple a machine which is badly configured. If a mismatch is found with the information in localinfo, the interface is reconfigured with the correct netmask and broadcast address. If a default route for packet forwadring is defined, this is checked against the static routing table and added if missing.

6. *cfengine* attempts to mount all defined file systems. This is achieved by opening a pipe to the shell mount command. Depressingly enough this is the most portable way of mounting file systems, since the system calls are completely screwed up. The information about the mounted file systems returned is stored in a list for eventual comparison with the list of file systems which *should* be mounted.

7. The list of mountables is parsed and file systems which are accessible to the current host are selected and checked aganist the list of already mounted file systems. Binary servers are defined separately in the binservers section of the program. Only binservers of the correct architecture are selected. Homeservers of any architecture may be mounted, but only if the host has permission (as described by homeservers). If a filesystem is wanting it is added to /etc/fstab or its equivalent for the given operating system type. All directories for the mounted file systems are created.

8. Any directories to be made are made.

9. Symbolic links are checked and installed if necessary. If a link exists and does not match the definition in the user program a warning is issued.

10. A check is made for the existence of the mail directory. If the host is a mailserver the following actions are ignored. If nothing is found in the list of mounted file systems, an entry is made in /etc/fstab or its equivalent.

11. A second attempt is made to mount all filesystems.

12. A check is made for the presence of "required" files and filesystems. These required objects are checked against internal variables which arbitrarily defined the minimum sensible size for a file or a filesystem. If the required object falls short of this size, a warning is issued. This could be caused for example by an NFS file system which did not mount. Since the directory exists no formal error is necessarily encountered.

13. Defined tidy items are tidied. Special tidy areas are tidied first. User home directories are then parsed and floating wildcards are deleted. A log of deleted files if stored for each user in /*.cfengine*.rm. Tidy descends into all subirectories in the user areas. In special tidy areas (those defined by an absolute path) directories which are hidden with an initial dot are not parsed. [4]

14. The system log of registered *setuid root* files is read in.

15. Any files to be disabled are renamed.

16. File systems are checked in the order in which they are defined. File systems marked with an R (recurse) flag are parsed fully. File systems marked wi th N (normal) do not have their subdirectories interrogated. See the sections above for a full desription of the file actions. If the action is "fix' and the owners and permissions do not match, the owner is set to the first item in the ownership lists for user and group and the permission is adjusted accordingl y. Files/directories to be touched and likchildren items are handled in the order in which they are defined.

17. The system log of registered *setuid root* files is appended and saved.

18. User defined scripts are executed in the order in which they are defined.

10 Portability

cfengine was designed to combat the problems of writing scripts which would work on a variety of systems. It is designed to be portable. A minimal amount of OS information is stored in the program itself in order to deal with system specific matters. In a later version it may be worth reading this in from configuration files to make it more easily extensible . This storage of information should be completely transparent to the end user, but will be of interest in porting the language to other platforms. The currently supported platforms are

[4]It is assumed that in system areas there maight be a reason for hiding these directories. For instance, /tmp/.X11 contains information about the open display. Deleting these files while open could result in the inability of programs to open the display.

soft class	Op. Sys.	Architecture	Release
sun4	sunos	sun 4?	4.1.*
ds	ultrix	risc	4*
hp	hp-ux	9000*	*
aix	aix	*	2
linux	linux	i?86	4*
osf	osf1	alpha	*

The interpreter has not been tested on *osf* or *linux* platforms (no machines are available!).

11 Known Bugs

The absence of a space before a closing parenthesis will sometimes cause the parser to complain of a syntax error. Under links: if there is no space between the arrow and the first pathname, no error will be given but the arrow is assumed to be part of the pathname. Caution with spaces!

A Simple Example program

Here is a small example program to show the principles. The output from running with the -v flag is shown at the end.

```
##############################################################################
#
#  System wide configuration file for the machines in fys.uio.no domain
#  that edb@fys.uio.no has control over. This file will be read by the
#  "configuration engine":
#
#  /fys/mutils/src/Cfengine/cfengine Version 2.0
#
#  Mark 5th September 1993
#
##############################################################################


localinfo:

   faculty    = ( mn )
   domain     = ( uio.no )
   sysadm     = ( drift@fys.uio.no )    # Only used in info from cfengine
   netmask    = ( 255.255.254.0 )
   timezone   = ( MET )

   mountpath = ( /<faculty>/<host> )
   homepat    = ( u? )                  # ( <faculty>-home-<host>? u1 u2 )

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

broadcast:

  any = (
        zeroes     # All the machines running this use zeroes to pad the
        )          # broadcast address.


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

defaultroute:

   alle = (
          fys-gw
          )

   tema = (
          innow-gw
          )

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

groups:
   teorfys       = ( +@fysikk-sun4-hosts -tema -aurora )
   struktfys     = ( tema )
   kosmisk       = ( aurora )
   faststoff     = ( gran sir hassel rogn alm linn selje vidje einer misteltein )

   allhservers   = ( anyon fidibus hope tema aurora hassel )
   allbservers   = ( anyon fidibus hope gran alm )
   usitsupport   = ( anyon fidibus )
   felles        = ( hope fidibus )
   alle          = ( +fysikk-hosts -tema )
   regnemaskiner = ( gluon elektron )
   vestfloyen    = ( hope fidibus wolfram )
   ostfloyen     = ( +fysikk-sun4-hosts aurora )    # :-)
   hpbinclients  = ( wolfram indium tin )
   hpbinservers  = ( hope gran )

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

shells:

   any =   (
           /bin/sh
           /bin/csh
           /usr/bin/ksh
           /usr/bin/sh5
           /local/bin/tcsh
           /local/bin/bash
           /local/bin/gnu/bash
           /local/gnu/bin/bash
           )

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

resolve:

   ostfloyen =
      (
      129.240.22.222
      129.240.2.3
      129.240.64.2
      )

   vestfloyen =
      (
      129.240.2.3
      129.240.22.222
      129.240.2.40
      )

   faststoff =
      (
      129.240.2.3
      129.240.22.222
      129.240.2.40
      )


   tema =
      (
      129.240.2.3
      129.240.64.2
      129.240.2.40
      )

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

homeservers:

  teorfys   = ( anyon fidibus )
  faststoff = ( hassel fidibus )
  felles    = ( fidibus hope anyon aurora tema gran )
  struktfys = ( tema )
  kosmisk   = ( aurora fidibus )
  wolfram   = ( fidibus hope )

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

binservers:

   teorfys.sun4   = ( gluino )
   any.sun4       = ( anyon )
   any.ds         = ( fidibus )
   any.hp         = ( hope )
   faststoff.hp   = ( gran )
   faststoff.aix  = ( alm )

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

mailserver:

   alle       = ( fidibus:/usr/spool/mail )             # /usr/spool/mail
   struktfys  = ( ulrik:/usr/spool/mail )

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

mountables:

   (
   anyon:/mn/anyon/u1              # check these against the hosts
   anyon:/mn/anyon/u2              # groups listed above in binservers/
   anyon:/mn/anyon/u3              # homservers and match a hostname in
   anyon:/mn/anyon/u4
   anyon:/mn/anyon/local           # an arbitrary field to figure out which
   anyon:/mn/anyon/fys             # host it belongs to.
   gluino:/mn/gluino/pc
   fidibus:/mn/fidibus/u1
   fidibus:/mn/fidibus/u2
   fidibus:/mn/fidibus/local
   fidibus:/mn/fidibus/fys
   hope:/mn/hope/fys
   hope:/mn/hope/local
   tema:/mn/tema/u1
   aurora:/mn/aurora/u1
   gran:/mn/gran/fys
   gran:/mn/gran/local
   hassel:/mn/hassel/u1
   alm:/mn/alm/fys
   alm:/mn/alm/local
   )

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

misc_mounts:

   sun4 = (
          mekker:/use/mekker/local /use/mekker/local ro
          info:/use/info/src       /use/info/src     ro
          )

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

makepath:

   allbservers = (
                 /mn/<host>/fys/public
                 )

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

links:

   any =
             (
             /fys                -> /<faculty>/<binserver>/fys
             /local              -> /<faculty>/<binserver>/local
             /usr/local          -> /local
             /etc/moduser.site -> /fys/mutils/bin/fys-reg       # used by ureg
             )

   sun4 =
      (
      /usr/lib/X11 -> /fys/mutils/X11
      )

   allbservers =
             (
             /public             -> /fys/public
             /local/bin/X11      -> /local/X11R5/bin
          )

arsen =   (
          /mn/arsen/fys      +> linkchildren
          )


aurora =  (
          /mn/aurora/fys     +> linkchildren
          )


elektron =
          (
          /mn/elektron/local +> linkchildren
          /scratch           -> /mn/elektron/scratch
          )

fidibus = (
          /usr/mail  -> /usr/spool/mail
          /fys/info  -> /mn/fidibus/fys/lib/gopherd/fysinfo
          )

gluon =
          (
          /mn/gluon/local +> linkchildren
          /scratch        -> /mn/gluon/local/scratch
          )

hpbinservers =
          (
          /fys/common     -> /mn/fidibus/fys/common
          /fys/info       -> /mn/fidibus/fys/info
          /fys/doc        -> /mn/fidibus/fys/doc
          /fys/src        -> /mn/fidibus/fys/src
          /fys/include    -> /mn/fidibus/fys/include
          /fys/modsim     -> /fys/modsim1.8
          /pc             -> /mn/fidibus/fys/pc
          /fys/proj/qgsm  -> /mn/fidibus/fys/proj/qgsm
          )

hp =      (
          /var/spool     -> /usr/spool
          /usr/spool/mail -> /usr/mail
          /usr/ucb/rdist -> /fys/mutils/archie/rdist/rdist
          /fys/man       -> /mn/fidibus/fys/man
          )

solaris = (
          /usr/spool      -> /var/spool
          /var/spool/mail -> /var/spool
          )
```

```
  teorfys = (
            /pc               -> /<faculty>/<binserver>/pc
            /public           -> /fys/public
            )


  hpbinclients =
            (
            /local            +> linkchildren
            )


 #
 # Next line is to link in USIT's packages.
 #

  usitsupport =
            (
            /local/bin         +> /local/perl/bin
            /local/bin         +> /local/TEX/bin
            /local/bin         +> /local/elm/bin
            )

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

tidy:

  allhservers = (
                core         >0
                *%           >3
                \#*          >1    # Backslash allows # so it isn't a comment
                *.dvi        >14
                *~           >3
                XtermLog.*   >3
                *.nfs        >0
                *.CKP        >1
                *.BAK        >1
                *.log        >14
                *.aux        >3
                .deleted     >2
                )

  allbservers =
               (
               /public/*%  >1
               /public/*~  >1
               )

  pcnfsd =
               (
               /var/spool/pcnfs/* >1
               )
  any =
                (
                /tmp/*            >1
                /etc/printcap.old >0
                )

  arsen =       (
                /core >0
                )

  regnemaskiner =
                (
                /scratch/* >2
                )

  fidibus =
               (
               /mn/fidibus/u2/anneho/*.o >7
               )

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

disable:

  any = (
        /etc/hosts.equiv
        /etc/nologin
        )

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

files:

   allbservers =
      (
      /fys/sbin                  -0002 R [root]       [0,1,3,mndrift]    fixall
      /local                     -0002 R [root,+use]  [0,1,3,tex,palsrc] fixall
      )

   anyon =
      (
      /public                    =0774 R [*]          [fysteori]         fixall
      )

   fidibus =
      (
      /fys/drift                 -0002+0770 R [root] [mndrift] fixall
      /fys/lib/gopherd/fysinfo   =0664      R [*]    [mndrift] fixall
      )

   felles =
      (
      /fys/proj                    -2002      R [*]    [*] fixall
      )

   gran =
      (
      /local/lib/idl               -0002      R [root] [*] fixall
      )

   allhservers =
      (
      home                         -2002      R [*]    [*] fixall  # all user dirs
      )

   any =
      (
      /etc/motd                    =0644      N [root] [*] fixall
      /.hushlogin                  =0644      N [root] [*] touch
      )

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

ignore:

   any =
      (
      !*
      /local/lib/gnu/emacs/lock/
      /local/tmp
      /fys/pc
      /fys/public
      /fys/info
      /fys/drift
      /fys/lib/gopherd
      /fys/ftp
      /fys/src
      )

   hpbinservers =
      (
      /fys/common
      /fys/doc
      /fys/info
      /fys/include
      /fys/pc
      /fys/src
      /fys/proj/qgsm
      )

   felles =
      (
      /fys/proj
      )
   gran =
      (
      /local/lib/idl
      )

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

required:

   any =     (
             /<faculty>/<binserver>/fys
             /<faculty>/<binserver>/local
             )

   hp =      (
             /local/bin/elm
             )

 #  teorfys = (
 #             /<faculty>/<binserver>/pc
 #             )


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

special:

   allbservers.sun4.scripts =
      (
      "/fys/lib/locate/updatedb"                      # GNU find. testing
      )


   allbservers.ds.scripts =
      (
      "/fys/lib/locate/updatedb"                      # GNU find. testing
      "/usr/etc/catman -w /local/man"
      "/usr/etc/catman -w /fys/man"
      "/usr/etc/catman -w /local/X11R5/man"
      )


 #
 # One day, catman will work even on HP machines
 #

   allbservers.hp.scripts =
      (
      "/fys/lib/locate/updatedb"         # GNU find. testing
      # "/etc/catman -w /local/man"
      # "/etc/catman -w /fys/man"
      # "/etc/catman -w /fys/mutils/man"
      )
```

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

scripts:                                     # should be called 'shell' ??

   any =
      (
      "/fys/mutils/cron/mweekly"
      )

   fidibus =
      (
      "/fys/mutils/cron/fidibus.daily"
      )
```

C Wrapper

Here is a simple wrapper program such as one might use to mail the system administrator. In this case the mail address is "admin@mydomain".

```
#!/local/bin/perl
#####################################################################
#
# Script wrapper, mails output if there is any
#
#####################################################################

$mail = 0;

$sysadm = "admin@mydomain";

$comm = join(" ",@ARGV);
@path = split(m;/;,$comm);
$suff = $path[$#path];
$tmpfile = "/tmp/mwrap.$$";
$subjectfile = "/tmp/mwrapsub.$$";

open (H,"/bin/hostname |") || die "mwrap: Can't get hostname";

 $hostname = <H>;

close(H);

open (SH,"$comm 2>&1 | ") || die "mwrap: Can't start shell\n";
open (OUT,">/tmp/mwrap.$$") || die "mwrap: Can't open workfile\n";

while (<SH>)
   {
   if ( /\S/ )
      {
      print OUT "$_";
      $mail = 1;
      }
   }


if ($mail)
   {
   open (SUB,">$subjectfile") || die "Cannot open $subjectfile\n";
   print SUB "Subject: $comm\n\n";
   print SUB "This message originates from host $hostname\n";
   print SUB "The full command issued was: $comm.\n\n";
   close(SUB);
   }

close(OUT);
close(SH);
```

B Verbose output

```
cfengine - Unix Configuration Engine - (C) M. Burgess 1993 V2.0

-------------------------------------------------------------------------

Host name is: anyon
Operating System Type is sunos
Operating System Release is 4.1.3
Architecture = sun4m


Using internal soft-class sun4 for host anyon

The time is now Mon Sep  6 09:19:45 1993


-------------------------------------------------------------------------


Defined Classes = ( any sun4 anyon teorfys allhservers allbservers usitsupport alle ostfloyen

Defined Binservers = ( anyon gluino anyon )

Defined home servers = ( anyon fidibus )

Defined wildcards to match home directories = ( u? )

Using mailserver fidibus:/usr/spool/mail

Local mountpoint is /mn/anyon/

Default route for packets 129.240.22.1
cfengine: editing /etc/resolv.conf
Interface name : le0
Address given by nameserver: 129.240.22.14
IP address on le0: 129.240.22.14
Netmask: 255.255.254.0
Broadcast address: 129.240.22.0
cfengine: default route is already set to 129.240.22.1

...
```

```
if ($mail)
   {
   system ("/bin/cat $subjectfile $tmpfile | /bin/mail $sysadm");
   }

unlink ("$subjectfile");
unlink ("$tmpfile");
```