Angewandte Mathematik und Informatik
Universität zu Köln

Report No. 99.379

The fully automatic installation of a Linux cluster

by

Mattias Gärtner, Thomas Lange,
Jens Rühmkorf

1999

Institut für Informatik
Universität zu Köln
Pohligstraße 1
D – 50969 Köln

Keywords: Linux, Debian, automatic installation, system administration

The fully automatic installation of a Linux cluster

Mattias Gärtner Thomas Lange Jens Rühmkorf

Institut für Informatik, Universität zu Köln

December 20, 1999

Abstract

We present a non interactive system, called FAI (Fully Automatic Installation), to install a Debian Linux operating system on a PC cluster. We take one or more virgin PCs, turn on the power and after a few minutes Linux is installed, configured and running on the whole cluster, without any interaction necessary. In addition, the configuration can be changed automatically on all Linux cluster nodes. Thus we have a scalable method for installing and updating a cluster with little effort involved. We use the Debian distribution and a collection of shell- and Perl-scripts for the installation process. Changes to the configuration files of the operating system are made by the tool cfengine.

Keywords: Linux, Debian, automatic installation, system administration

Contents

1 Motivation

Have you ever performed identical installations of an operating system several times? Would you like to be able to install a Linux cluster with dozens of nodes single handedly?

Repeating the same task time and again is boring – and will surely lead to mistakes. Also a whole lot of time could be saved, if the installation were done automatically. An installation process with manual interaction does not scale. But clusters have the habit of growing over the years. Think long-term rather than plan only just a few months into the future. When we received hardware for our Linux cluster – 1 server and 16 clients – we decided to do a fully automatic installation of the cluster. It was obvious that it would take some time to get things to work, but also that we would save much time in the future. In the past, we had had much experience with the installation of the Solaris™ operating system on SUN SPARC hardware. Solaris has an automatic installation feature called JumpStart™[13]. In conjunction with the auto-install scripts from Casper Dik[16] we saved a lot of time not only for every new SUN computer, but also for reinstallation of existing workstations. For example, we had to build a temporary LAN with four SUN workstations for a conference, lasting only a few days. We took these workstations out of our normal research network and set up a new installation for the conference. When it was over, we simply integrated the workstation back into the research network, rebooted just once, and after half an hour, everything was up and running as before. The configuration of all workstations was exactly the same as before the conference, because everything was performed by the same installation process. We also used the automatic installation for reinstalling a workstation after a damaged hard disk had been replaced. It took two weeks until we received the new hard disk but only a few minutes after the new disk was installed, the workstation was running as before. And this is why we chose to adapt this technique to a PC cluster running Linux.

The choice to use Debian Linux [10] was made, since some experience with this distribution had been gathered. It was not possible to predict, if a different Linux distribution would support this kind of installation better.

The Linux cluster will be a platform for the development of software for parallel methods for the satisfiability problem (SAT) and the CATS-project "Computer Aided Tram Scheduling" as well as Also the design and implementation of efficient parallel adaptive multigrid methods is an object of research.

2 Overview

2.1 Hardware

The following hardware was purchased for the linux cluster:

Server: named lichtenstein

- Asus P2B-DS Mainboard
- 2 × Intel Pentium II 400 Mhz

- 512 MByte SDRAM (PC100)
- 3Com FastEtherlink XL, 10/100 Mbit, 3c905B chip
- Adaptec AIC-7890/1 Ultra2 SCSI host adapter
- 2 × 9 GByte hard disk, IBM DDRS-39130D
- ATI Xpert-Work, AGP, 8 MB graphic card
- TEAC CD-532S, 32 x speed SCSI-CDROM
- 1.44 MB floppy disk

16 Clients: named roy01 to roy16, each equipped with

- Gigabyte 6BXD Mainboard
- 2 × Intel Pentium II 400 Mhz
- 256 MByte SDRAM (PC100)
- 3Com FastEtherlink XL, 10/100 Mbit, 3c905B chip
- 4,3 GByte hard disk, Western Digital Caviar WDC AC24300L
- S3 Virge DX, 4 MB graphic card

Switch: 24 ports 10/100Mbit, Cisco Catalyst C2924-XL

The overall cost for this hardware was about 30.000 Euro (purchased at the end of 1998). All clients share one keyboard and one monitor through a tree of manual keyboard and monitor switch boxes. Neither keyboard nor monitor are needed for an automatic installation.

2.2 Requirements and preliminary work

All that is needed for a fully automatic installation is a server providing BOOTP, NFS and TFTP services. TFTP is only needed if the system is not booted from floppy, but via the network card. A server running Linux is recommended but not mandatory. A running Linux system is required in order to build new kernels. Access to all Debian packages via NFS (mostly on the local NFS server) is needed.

The computer to be installed – called *install client*[1] or *client* for short – should boot from its network card or from floppy. Time should then be invested to adjust the configuration to local needs. Before booting the clients, four tasks must be performed:

- Set up BOOTP, NFS (and TFTP) services on the server.
- Create a kernel image that recognizes the network card and could mount its root filesystem via NFS from the server. There is already a kernel image available, that should work with most hardware.
- Create the root filesystem on the server. This is used only during the installation process, not afterwards. This applies to all clients and is created by a simple script.
- Define how the clients should be installed (called *configuration*). The configuration consists of:

[1] An install client can also be installed as a server

- partition tables for the local disks
- mount information for local filesystems
- names of software packages to be installed
- information on changes and supplements for the operating system

Most of the work is spent in creating the first configuration. Once a suitable configuration exists, a new computer with slightly different equipment would usually require no change to the configuration. If the requirements are however different, changes to the existing configuration usually require little effort. It is also possible to make changes on a running system rather than performing a complete new installation.

2.3 Overview of the installation sequence

During an installation the following steps are performed:

1. The client boots via the network or from floppy, starting a fully functional Linux operating system *without* using the local disk.
2. The local hard disks are partitioned and empty filesystems are created on all partitions, if desired.
3. The selected software packages are installed.
4. The changes to the configuration of the operating system are carried out.
5. The install client reboots from its local disk and installation is completed.

At present, it is safer to reboot a second time, as a script is executed during the first booting from local disk. This is done automatically. The first step is also very useful if parts of the local hard disk are damaged and a backup must be performed.

3 Setting up the server

3.1 Preconditions

A Debian Linux distribution is used to install Linux on the clients together with cfengine and some scripts. The server is called *lichtenstein* and our clients are named *roy01* through *roy16*.

Currently our clients are installed as dataless clients, mounting `/usr` and `/home` from the server. All computers are connected directly to a switch and are using one class C subnet. We are using NIS [6](Network Information Service, formerly known as Yellow Pages YP) to distribute data for passwd, hosts, netgroup and other files. Our NIS Server is a SUN Enterprise 450 and all Linux hosts are NIS clients. Setting up NIS will not be explained in this paper, as it is not needed for an automatic installation. Almost all files for the fully automatic installation are located under `/files/install`. The three main directories used for the installation are:

`fai/`	about 5 MB,	all configuration files
`root/`	about 30 MB,	untar'ed file base2_1.tgz
`debian/`	about 1.2 GB,	Debian 2.1 distribution with packages main, contrib, non-free

Separate directories for each client are not required. All three directories are exported read only. The disk size needed is mainly determined only by the size of Debian packages. Here is an extract of the directory tree of FAI, showing the main parts:

```
lichtenstein[/files/install]# tree -d fai
fai/
|-- fai_scripts
|-- class
|-- disk_config
|-- package_config
|-- scripts
|-- doc
|-- etc
|-- files
|-- kernel
```

All base FAI scripts are located in the subdirectory `fai_scripts`. The subdirectory `class` contains all scripts and files for defining classes for the clients. The configuration for partitioning the hard disk and mounting local filesystems are stored in `disk_config`. Information for software packages can be found in `package_config`. The scripts that are executed at the end of the installation are stored in subdirectory `scripts`. Templates for files that are copied onto the clients are located in `files`. The Subdirectories `etc` and `kernel` contain files for NIS and BOOTPD and the scripts for building different kernels. Finally, `doc` contains some documentation.

For each client we have to define the ethernet and IP address and make an entry for netgroup. We add this data into the NIS tables. Without using NIS these entries are made in `/etc/ethers`, `/etc/hosts` and `/etc/netgroup`.

3.2 Network daemons running on the server

To enable TFTP and BOOTP on a server, the following lines are usually added to the file `/etc/inetd.conf` or, should they already exist, they are uncommented:

```
tftp   dgram udp wait nobody /usr/sbin/in.tftpd in.tftpd /tftpboot/
bootps dgram udp wait root   /usr/sbin/bootpd  bootpd -t 120
```

After changing this file, *inetd* is instructed to reread its configuration file.

```
lichtenstein[~]# killall -v -HUP inetd
Killed inetd(196)
```

Debian Linux contains a program *killall*, which kills processes by name. If this is not available, simply use `kill -HUP <pid of process>`. Normally, BOOTP requests are broadcast only within a subnet. If the clients are connected to a different subnet than the BOOTP server, the router configuration should be altered, or a BOOTP gateway (see bootpgw(8)) should be used to forward the requests to the BOOTP server.

To enable NFS service, *rpc.nfsd* and *rpc.mountd* daemons must be started. Debian does this by executing the script `netstd_nfs` (Debian version 2.0) or `nfs-server` (since Debian version 2.1), which are located in `/etc/init.d`. The file `/etc/exports` controls which directories can be mounted by which hosts. The following directories must exported, so the clients are able to mount them:

```
lichtenstein[~]> cat /etc/exports
/usr                    @linux-cluster(ro,no_root_squash)
/files/install/root     @linux-cluster(ro,no_root_squash)
/files/install/fai      @linux-cluster(ro,no_root_squash)
/files/install/debian   @linux-cluster(ro,no_root_squash)
```

The netgroup `@linux-cluster` contains all install clients and is distributed via NIS. A netgroup can also be defined in the file `/etc/netgroup`[2]. The contents of the exported directories are described later in detail. After changing `/etc/exports`, the mount daemon is instructed to reload its configuration file in the same way as *inetd*.

```
lichtenstein[~]# killall -v -HUP rpc.mountd
Killed rpc.mountd(23870)
```

3.3 Creating the root filesystem for clients

With Debian, it is very easy to create the root filesystem, which is mounted read only by the clients during the installation process. Debian supports a "base" root filesystem, which includes the essential packages which are absolute required. For Debian 2.1 (also called slink) this is `base2_1.tgz`, which can be found in `slink/main/disks-i386/current/`. The script `create_client_root.sh` (sources in the appendix) extracts the files of the tar archive. As a next step, some symbolic links to `/tmp` are made for the files which must be writable. A ramdisk allows writing to files located in `/tmp`. Only one missing binary and a script must be copied into the root filesystem. The binary `/sbin/bootpc` [14] is a BOOTP client, which receives data from a BOOTP server for a client and prints it. This is part of the *netstd* package. The original script `/etc/init.d/rcS` is replaced by the new script, which performs the installation. When a client has booted its kernel, `rcS` is the first script which is executed by the init process. This script controls the sequence of the installation.

3.4 Debian software packages

Each client receives the software packages that will be installed over the network. If several clients are to be installed, this could produce a great amount of network traffic. Therefore

[2] see netgroup(5) for more information

a local copy of all needed Debian software packages is recommended. We are using a local copy stored in `/files/install/debian` on the server. This is a copy of Debian 2.1 retaining the directory structure of the Debian distribution. Figure 1 shows an extract of the tree structure of the Debian distribution.

```
lichtenstein[~]> tree /files/install/debian -d
/files/install/debian
`-- dists
  |-- Debian2.1r2 -> slink
  |-- slink
  |   |-- contrib
  |   |   |-- binary-all
  |   |   |   |-- admin
  |   |   |   |-- base
  |   |   |   `-- x11
  |   |   `-- binary-i386
  |   |       |-- admin
  |   |       |-- base
  |   |   |   `-- x11
  |   |-- main
  |   |   |-- binary-all
  |   |   |-- binary-i386
  |   |   |-- disks-i386
  |   |   |   |-- 2.1.9-1999-03-03
  |   |   |   `-- current -> 2.1.9-1999-03-03
  |   |   `-- upgrade-2.0-i386
  |   `-- non-free
  |       |-- binary-all
  |-- stable -> slink
```

Figure 1: Directory structure for Debian (extract)

3.5 Other exported directories

The `/usr` partition of a linux host must also be exported. It is mounted during the installation, so all needed binaries are available. The subdirectory `/files/install/fai` contains all configuration information and is described in section 6.

3.6 Building BOOTP Configuration

If BOOTP has been setup on the server, it must be fed with the necessary data. As an example, here is our `/etc/bootptab`:

/etc/bootptab

```
.global.prof:\
        :ms=1024:\
        :sa=lichtenstein:\
        :hd=/tftpboot/:\
        :hn:bs=auto:\
        :rp=/files/install/root:\
        :ts=rubens:\
        :T170="134.95.9.100:/files/install/fai":
        :T171="install":\
        :sm=255.255.255.0:\
        :gw=134.95.9.254:\
        :dn=informatik.uni-koeln.de:\
        :ds=134.95.9.136,134.95.100.209,134.95.100.208:\
        :ys=rubens:yd=informatik4711.YP:\
        :nt=time.rrz.uni-koeln.de,time2.rrz.uni-koeln.de:

# T170 is used for the location of the fai directory
# T171 "install" means do the installation, else execute a shell

roy01:ha=0x00105a270b29:bf=roy01:tc=.global.prof:
roy02:ha=0x00105A270c08:bf=roy02:tc=.global.prof:
```

In this example, the BOOTP configuration is identical for all clients. It is nevertheless possible to define different NIS servers for different hosts, or different domain name servers for certain hosts. Using different directories for the FAI configuration (T170) is not recommended, because we use classes within FAI to specify different configurations. The root path should also be identical for all clients, in order to save disk space. Clients can use the same root directory simultaneously, because they do not have write permission for it.

With the option `hn`, the client's hostname is sent to the client instead of the numerical IP-address. Option `ms` is needed, because the configuration exceeds a certain size. Setting `bs=auto` prevents defining the size of the boot file, which is the concatenation of options `hd` and `bf` e.g., for *roy01* the filename is `/tftpboot/roy01`. The values of `sm` and `gw` are used for the booting process. The network card will use them to configure itself correctly. The following variables are later used during the configuration of the operating system:

ys Name of NIS server

yd Name of NIS domain

ts Time server address list

nt NTP (network time protocol) server list

dn Domain name that is used in resolv.conf

ds Domain name server address list

sa TFTP server address

rp Root path to mount as root

T170 This is a generic tag. It is used for the location of the FAI directory.

T171 This generic tag defines if an installation should be performed or if a shell will be executed.

There are two generic tags – T170 and T171. The choice of numbers are random. This feature may be used in future to pass more data to the clients. See figure 2 (on page 17), for how this data is passed to the client. The manual pages of bootptab(5) contain more information.

4 Booting clients

4.1 Creating a boot floppy

There are two methods for booting the clients. The computer can boot from its network interface card (NIC) to receive the boot image via `BOOTP/TFTP`, or an appropriate kernel is loaded from a floppy. Booting from a network card is described is section 4.3.

Should booting take place from floppy, creating a boot floppy is very easy for most network cards. The file `bzImage.install` must be simply copied onto a floppy.

```
# dd if=/files/install/fai/kernel/bzImage.install of=/dev/fd0
```

This is a bzImage (kernel version 2.0.36) with most device drivers compiled into the kernel and the root device is changed with *rdev(8)* from `/dev/hda1` to `0x00ff`. The configuration for compiling this kernel is saved into `bzImage.install.config`. The boot floppy can now be tested (see section 4.2).

If this bootfloppy does not work, a new kernel has to be compiled. In order to compile this installkernel, the BOOTP option has to be enabled, so the kernel will mount the root filesystem via NFS. For kernel versions up to 2.1 series, these options are located in menu `NFS filesystem support` and are called `Root file system on NFS` and `BOOTP support`. In kernels newer than 2.1 activate `Networking options -> IP: Kernel level autoconfiguration` and `Filesystems -> Network File Systems -> NFS filesystems support -> Root file system on NFS`[3]. The options *ramdisk*, *proc filesystem* and *rtc* (real time clock) support are also required, which will mostly be enabled by default. The option *initrd* (initial RAM disk support) must not be enabled. After compiling the kernel, the default root device should be changed in order to determined by BOOTP. The following commands are used to change it and to write the kernelimage onto a floppy:

```
lichtenstein[~]# cd /usr/src/linux/arch/i386/boot
lichtenstein[~]# rdev bzImage
Root device /dev/hda1
lichtenstein[~]# mknod /dev/boot255 c 0 255
```

[3]Thank to Jakob Flierl for this hint. See http://www.luga.de/~flierl/diskless_suse

```
lichtenstein[~]# rdev /dev/fd0 /dev/boot255
lichtenstein[~]# rm -f /dev/boot255
lichtenstein[~]# rdev bzImage
Root device 0x00ff
lichtenstein[~]# dd if=bzImage of=/dev/fd0
```

The first `rdev` call shows the current root device for the kernel image. Then a temporary device is created and set with the second call of `rdev`. Finaly we copy the kernel to the floppy.

4.2 Booting from floppy

The floppy is tested by booting the computer from it. Here are some of the messages for the client *roy01* which is booting without errors:

```
Loading.................
Uncompressing Linux...done.
Now booting the kernel
.
.
Linux version 2.2.10 (root@faiserver) (gcc version 2.7.2.3)
#11 SMP Thu Dec 16 12:33:01 MET 1999
Processor #0 Pentium(tm) Pro APIC version 17
Processor #1 Pentium(tm) Pro APIC version 17
Processors: 2
Detected 398944669 Hz processor.
Console: colour VGA+ 80x25
.
.
Partition check:
 hda: hda1 hda2 hda3 hda4 < hda5 hda6 hda7 hda8 >
Sending BOOTP request...... OK
Root-NFS: Got BOOTP answer from 134.95.9.100, my address is 134.95.9.101
Root-NFS: Got file handle for /files/install/root via RPC
```

These are the messages seen during successful booting. If the client receives no response from a BOOTP server, the following message appears:

```
Sending BOOTP request.............. timed out!
```

This means that the boot floppy is OK, but the computer can not connect to a BOOTP server. If the network card is not recognized by the kernel, the following error message is printed:

```
Root-NFS unable to open at least one network device
```

Then a new kernel with support for the installed network card has to be compiled.

4.3 Booting from network card

We distinguish two kernel. One kernel, also called the *install kernel*, is used during the installation process. The other kernel, we named it *cluster kernel*[4], is used for normal operation when the client is booting from the local disk. These kernels do not need to be identical. Kernel version 2.0.36 (the default kernel for Debian 2.1) is currently used during the installation, and version 2.2.10 is used when the clients have booted from local disk, since we are using clients with two CPU's each, and the newer kernels better support SMP (Symmetric Multi Processing).

For administrative purposes, booting from network card (NIC) is much more suitable than booting from floppy. In order to use this boot method, a boot ROM that is able to communicate with a `BOOTP` server to receive communication-related configuration values such as network addresses and which is capable of communicating with a `TFTP`-server to obtain a boot image must be obtained. Furthermore, it must be guaranteed that the transmitted boot image is executed properly in terms of what the boot ROM expects in a boot image. Our boot ROM failed to execute a bzImage, which we had created to boot from floppy, so we had to find another solution.

Booting Linux via network card is be done by using either Netboot [8], Etherboot [9] or NILO[5]. The first two programs are capable of creating a boot ROM binary (which must be programmed onto a ROM) and a corresponding `TFTP` boot image which includes a kernel image. Some tools, exist that help test a boot ROM image for example.

The advantage of Netboot is its ability to emulate just enough of a `DOS` environment such that unmodified `DOS` packet driver binaries (these are usually provided with the NIC) can be used for building a boot ROM. Etherboot, on the other hand, creates smaller boot ROM images; the compressed versions will fit in 8 KB (all NIC's should support this size). Also Etherboot does autoprobing of the hardware addresses, while Netboot only does autoprobing as long as the packet driver supports this feature. However, the boot ROM binary one of these programs created will most likely have to be burnt onto an appropriate PROM, but if care it taken in the choice of hardware to be bought, there are no problems to get the network card to work. For our Linux cluster, however, we use neither Netboot, Etherboot, nor a PXE-compliant boot ROM, but use the boot ROM by Lanworks Technologies that comes with the 3Com-FastEtherlink XL 3c905B NICs. Since most proprietary solutions are based upon Intel's PXE-specification [15, 5] – which is supported by the Lanwork ROM currently is use, as well – this seems to be a rather unusual workaround. Since it works well for our purposes, we did not find the need to change the procedure. If it is clear that either Netboot or Etherboot, or a solution based on a proprietary PXE-compliant boot ROM will be used, this part can be disregarded.

3Com usually equips its NICs with the Managed PC Boot Agent (MBA) ROM by Lanwork Technologies. MBA's version v3.10 worked well after we configured the ROM as follows (press `Ctrl+Alt+B` during boot up):

[4]Since we use it for our Linux cluster

[5]While this document is written, developments are made to the NILO project[11]. Its goal is to provide network booting ROM - based on Linux network adaptor drivers - which support the Intel PXE specification.

```
Managed PC Boot Agent (MBA) v3.10
(C) Copyright 1998 Lanworks Technologies Co. a subsidiary of 3Com Corporation
All rights reserved.
================================================================================
                            Configuration

Boot Method:                TCP/IP
Protocol:                   BOOTP
Default Boot:               Network
Local Boot:                 Disabled
Config Message:             Enabled
Message Timeout:            3 Seconds
Boot Failure Prompt:        Wait for key
================================================================================
  Use cursor keys to edit: Up/Down change field, Left/Right change value
  ESC to quit, F9 restore previous settings, F10 to save
```

In order to build a file that can be loaded via TFTP from the NIC's and that is able to boot the kernel properly, we use the tools that are enclosed in the ROM package. The tools require a DOS-formatted disk for booting the Linux kernel. The TFTP boot image we are using is created with this disk. We use *syslinux(1)* to build such a disk:

```
lichtenstein[~]# superformat /dev/fd0
Verifying cylinder 79, head 1
mformat -s18 -t80 -h2 -S2 -M512  a:
lichtenstein[~]# mount -t msdos /dev/fd0 /floppy/
lichtenstein[~]# cat syslinux.cfg
TIMEOUT 40
PROMPT 0
DEFAULT bzImage
APPEND root=/dev/nfs
lichtenstein[~]# cp syslinux.cfg /floppy/
lichtenstein[~]# cp bzImage /floppy/
lichtenstein[~]# umount /floppy/
lichtenstein[~]# syslinux /dev/fd0
```

If the BIOS causes trouble booting from this disk, the command `syslinux -s /dev/fd0` is used. Afterwards `imagegen`, a tool from the MBA-software, is used on a standard MS-DOS-computer to build the TFTP boot image (`a:` denotes the disk just created using `syslinux`, `tftpboot.img` denotes the TFTP boot image which is to be created).

```
C:\> imagegen a: tftpboot.img
```

The `imagegen` command is invoked only once. The image file that is created by `imagegen` is exactly 1024 bytes larger than a regular DOS-disk. Therefore all we need are these first 1024bytes and a working `syslinux` boot disk to create a valid TFTP boot image:

```
lichtenstein[~]# dd if=tftpboot.img of=first_block bs=1024 count=1
1+0 records in
1+0 records out
lichtenstein[~]# cp first_block new_tftpboot.img
lichtenstein[~]# dd if=/dev/fd0 of=new_tftpboot.img bs=1024 seek=1
1440+0 records in
1440+0 records out
lichtenstein[~]# cp new_tftpboot.img /tftpboot/installimage
lichtenstein[~]# chmod a=r /tftpboot/installimage
```

This is all it takes to create a `TFTP` boot image. There is a shell script `kernel2image.sh` doing exactly the described operation by mounting an image of an empty `DOS`-disk as loopback device. We distinguish between 'install'-booting[6] and 'normal'-booting[7] our Linux cluster. Thus each client has a link to one of the two different boot images in `/tftpboot/`.

```
lichtenstein[...fai/kernel]# ls -l /tftpboot/
total 2896
-r--r--r--  1 root root      1475584 Aug 18 15:23 clusterimage
-r--r--r--  1 root root      1475584 Aug 18 00:46 installimage
lrwxrwxrwx  1 fai  linuxadm       12 Aug 20 15:47 roy01 -> installimage
lrwxrwxrwx  1 fai  linuxadm       12 Aug 20 15:03 roy02 -> installimage
```

Boot time arguments are passed to the kernel using the configuration file syslinux.cfg. Using such an "append" parameter, we let the kernel, which is loaded by the `TFTP` boot image, boot either from root device `/dev/hda1` or `/dev/nfs`[8]. Finally, this is how the `TFTP` boot images using the script `kernel2image.sh` were actually build for a IDE hard disk:

```
lichtenstein[...fai/kernel]# kernel2image.sh clusterimage bzImage /dev/hda1
generate:
       tftp boot image "clusterimage" from
       kernel "bzImage" with
       append-param. "root=/dev/hda1"?
type ctrl-c to abort, return to continue

step 1/6:  generate temporary DOS-disk
step 2/6:  copy kernel to disk
step 3/6:  create syslinux.cfg on disk
step 4/6:  install syslinux
step 5/6:  copy MBA's imagegen-loader to tftp boot image
step 6/6:  append DOS-disk to tftp boot image
```

[6] Using `/files/install/root` as root filesystem

[7] Using the root filesystem from `/dev/hda1`

[8] To be exact, `/dev/nfs` is not really a device, but rather a flag to tell the kernel to get the root filesystem via the network.

```
1440+0 records in
1440+0 records out
+++ tftp boot image "clusterimage" generated. +++

lichtenstein[...fai/kernel]# kernel2image.sh installimage bzImage /dev/nfs
.
.
+++ tftp boot image "installimage" generated. +++
```

Now all information to test if the clients can boot with the selected method is available. Setting `T171="Xinstall"`, the client boots but does not perform the installation.

5 The installation process

This section explains the installation process in detail. The host *roy01* is used in our examples. After uncompressing and successfully booting the kernel, the root directory is mounted (for boot messages see section 4.2) and the first process (*init*) is spawned. The file `/etc/inittab` defines that `/etc/init.d/rcS` is the first process started by *init*. Since the client mounts its root filesystem `/files/install/root` from the server, it in fact executes `/files/install/root/etc/init.d/rcS`, which is the new script for the fully automatic installation. A copy of the script resides in `/files/install/fai/fai_scripts`. The following steps are performed in `rcS`:

1. initialize Linux
2. setup FAI
3. define classes
4. format local disk
5. install software packages
6. call cfengine or other scripts
7. save log files
8. reboot

We now describe the operation of this script.

5.1 Init and setup routines

First, the subroutine *fai_init* is called.

fai_init

```
1   fai_init() {

        PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/sbin\
        :/usr/local/bin:/fai/fai_scripts
```

```
 5      export PATH
        umask 022

        mount -n -t proc proc /proc
        cat /proc/kmsg >/dev/tty4 &
10      [ -x /sbin/update ] && update
        create_ramdisk /dev/ram0
        > /tmp/FAI_INSTALLATION_IN_PROGRESS
        trap 'exec sh' 2
        dmesg > /tmp/dmesg.log
15
        echo ""
        echo "$0: starting fully automatic installation FAI ..."
        echo "Press ctrl-c to interrupt installation process and to get a shell"

20      # TODO: if timeout for bootpc exit installation
        # define all bootpc information as variables
        bootpc | sed -e 's/^/export /' > /tmp/bootpc.log
        . /tmp/bootpc.log
        hostname $HOSTNAME
25
        if [ "$T171" != "install" ]; then
            echo /etc/bootptab: T171 != install. Not performing FAI installation.
            exec sh
        fi
30  }
```

This subroutine mounts the proc filesystem first, since it contains information on the hardware and the running system (see manuals of proc(5)). Line 9 redirects all kernel messages to a virtual console that can be viewed by typing `Alt-F4`. Then the update daemon is started, flushing the filesystem buffers at a regular interval. In line 11, the subroutine *create_ramdisk* is called to create a ramdisk on `/dev/ram0`. The ramdisk is mounted on `/tmp`, where all writable files including all log files are stored. Line 13 enables the feature to interrupt the installation process and to execute a bash shell by typing `ctrl-c`. Debugging is therefore possible, should functions not work as expected. After displaying a few messages, *bootpc* is called. This BOOTP client program receives all data from the BOOTP server and stores it in a temporary file. Using the simple *sed* script, the syntax of the output is changed and used as a normal shell script. Figure 2 shows the file for client *roy01*. The script `rcS` sources this file and defines all the variables. In co-operation with the generic tags of BOOTP, a lot of information is passed to the client. If *bootpc* prints an error message, we check `/etc/bootptab` or start the BOOTP daemon with debug options enabled. The first step is finished with the setting of the hostname.

The procedure *fai_setup* mounts the configuration directory and reads the global configuration `fai.conf` (see appendix, page 32). All variables with prefix `FAI_` are defined in this file. Before mounting `/usr` from the server, only a few executables in `/files/install/root` are available. After mounting, all programs and most libraries are available including *rdate*,

```
lichtenstein[~]# cat ~fai/roy01/bootpc.log
export SERVER='134.95.9.100'
export IPADDR='134.95.9.101'
export BOOTFILE='/tftpboot//roy01'
export NETMASK='255.255.255.0'
export NETWORK='134.95.9.0'
export BROADCAST='134.95.9.255'
export GATEWAYS_1='134.95.9.254'
export GATEWAYS='134.95.9.254'
export ROOT_PATH='/files/install/root'
export DNSSRVS_1='134.95.9.136'
export DNSSRVS_2='134.95.100.209'
export DNSSRVS_3='134.95.100.208'
export DNSSRVS='134.95.9.136 134.95.100.209 134.95.100.208'
export DOMAIN='informatik.uni-koeln.de'
export SEARCH='informatik.uni-koeln.de uni-koeln.de'
export YPSRVR_1='134.95.9.10'
export YPSRVR='134.95.9.10'
export YPDOMAIN='informatik4711.YP'
export TIMESRVS_1='134.95.9.10'
export TIMESRVS='134.95.9.10'
export NTPSRVS_1='134.95.100.209'
export NTPSRVS_2='134.95.170.8'
export NTPSRVS='134.95.100.209 134.95.170.8'
export HOSTNAME='roy01'
export T170='134.95.9.100:/files/install/fai'
export T171='install'
```

Figure 2: bootpc.log for roy01

which is then executed to set the local time. However, the time may be shown for a different timezone. It is set correctly at the end of the installation process.

fai_setup

```
1   fai_setup() {

        # generic tag 170 (bootptab) used for location of fai directory
        export FAI_LOCATION=$T170
5       mount -o ro $FAI_LOCATION /fai
        # read global config for fai
        if [ -r /fai/fai.conf ]; then
            echo  mounting FAI directory from $FAI_LOCATION
            . /fai/fai.conf
10          echo $FAI_VERSION
            echo ""
        else
            echo mounting $FAI_LOCATION failed
            echo "or can't read /fai/fai.conf"
```

```
15          echo "Can't start fully automatic installation."
            sh
        fi

        # after mounting /usr, we have everything needed
20      mount -o ro -n -t nfs ${FAI_NFSSERVER}:/usr /usr &&
            echo /usr mounted from ${FAI_NFSSERVER}

        rdate ${TIMESRVS_1}
    }
```

5.2 Defining classes

The subroutine *define_classes* is then called. The variable *$classes* contains a list of all the classes that are defined for the client. We also say "the client belongs to these classes". Classes control how a client will be installed. This feature is described later in section 6. The subroutine *defined_classes* calls all scripts in `fai/class`, whose file name match the pattern `S[0-9]*.{sh,pl,source}` (filenames start with an uppercase `S` follow by a digit and any other character ending in `.sh`, `.pl` or `.source`) and which are executable. These scripts are called in alphabetical order and print the names of the classes to standard output to define them. Files with postfix `.source` need not define classes, but are used to define variables for cfengine.

define_classes

```
 1  define_classes() {

        cd /fai/class

 5      # alphabetical sort is important
        for f in `ls S[0-9]*.{sh,pl,source}` ; do
            if [ -x $f ] && [ -f $f ]; then
            [ -n "$verbose" ] && echo executing $f

10          case $f in

                *.pl) newclasses=`perl $f </dev/null` ;;

                *.sh) newclasses=`sh $f </dev/null`   ;;
15
                # source files, which can set variables
                *.source)
                [ -n "$debug" ] && set -v
                . $f </dev/null
20              [ -n "$debug" ] && set +v
                newclasses=
                ;;
```

```
            esac
            [ -n "$debug" ] && echo "   newclasses= $newclasses"
            export classes="$classes $newclasses"
            fi
        done
    .
    .
    }
```

5.3 Partitioning disks

After the classes are defined, the main installation part starts. The local disks are configured by calling the script `setup_harddisk.pl` (located in `/fai/fai_scripts`). The script searches for a disk configuration file in `/fai/disk_config`, whose name is a class to which the client belongs. All definitions for the disk layout must be stored in one file. The local disks are partitioned, and the script creates empty filesystems on these partitions by default. Moreover, the data on a partition can be preserved, if desired. The partitions are mounted on `$FAI_ROOT` according to the predefined mount points. The disk configuration for *roy01* is stored in the file `4GB`, because the script `S07disk.pl` (see page 26) defines this class for *roy01* and no other configuration file with name roy01 exists.

```
lichtenstein# cat 4GB
# disk configuration for one disk with 1000-4000kb

# <type> <mountpoint> <size in mb> [mount options]     [;extra options]

disk_config hda

primary  /             30          rw,errors=remount-ro ;-c
logical  swap          200         rw
logical  /var          50-200      rw
logical  /usr          70          rw
logical  /tmp          100-150                          ;-m 0
#logical  /scratch     0-          rw,nosuid            ;-m 0 -i 50000
logical  /scratch      preserve9   rw,nosuid            ;-m 0 -i 50000
```

It is possible to define the size, the mount point, the mount options and extra options (mostly for mke2fs) for each partition. A new filesystem is created on each partition by default. However, the size and the data of a partition can also be preserved. Preserving data is done by specifying the size as `preseve<no>`, whereas `<no>` is the device number of the partition that must remain unchanged. If an interval is defined for several partition sizes, the script tries to maximize these sizes, preserving the ratio between them. A detailed description can be found in `fai/doc/README.disk_config`.

5.4 Software installation

After mounting the disks, the Debian software packages are installed. Debian uses a "base" tar file which includes all required software packages. It is the same tar file that is used for creating the root filesystem in `/files/install/root/` on the server. The script `install_base_root.sh` mounts the directory containing all Debian packages to `/fai/debian/` and extracts files from the base file. After these packages are installed, the other necessary packages are installed on the client. For this, we use the script `install_packages.pl`, which reads all configuration files from `/fai/package_config/` matching a class name, is used. Client *roy01* only installs software defined in file `ROY`, because it is a dataless client, mounting most of the software from the server. Here are two examples for software configuration files:

```
lichtenstein[...fai/package_config]> cat ROY
PACKAGES install
netstd lpr pciutils sysutils time strace ldso
tcsh tcsh-i18n less cfengine
psmisc psutils
cron mpich

lichtenstein[...fai/package_config]> cat COMPILE
# packages for developing software
PACKAGES install
cpp bin86 binutils m4 make
libc6-dev libg++2.8.2 libstdc++2.9-dev
g++ gcc gdb libstdc++2.9
flex g77 byacc cvs
```

The script uses the Debian command *apt-get(8)*. This new command-line tool for handling packages – like *dpkg(8)* – is currently under development. Therefore, with a new versions of *apt-get*, new features will be added, all of which will make this part of the automatic installation more comfortable. The configuration file starts with the string `PACKAGES` followed by an *apt-get(8)* command. Currently only the command `install` is used, but there are some other commands like `remove` or `upgrade`.

Currently *apt-get* fails during the installation of some software packages. Installing a Debian package comprises several steps. It is important to realize that installing a package also includes unpacking and configuring. During the configuration, an existing postinstall script (see `/var/lib/dpkg/info/*.postinst`) for this package is called, which may execute any command. This is a problem for the fully automatic installation, since a `chroot $FAI_ROOT` is performed during installation via *apt-get*. This means that some parts of the postinstall scripts fail to get their current working directory, or that daemon processes cannot be started or stopped. The main problem, however, are manual input requests by a post install script. This has to be suppressed, since we want automatic installation without any manual user interaction. Nevertheless it was possible to install the software packages

without any interaction. For this purpose `yes ""| dpkg --configure -a` is called after the installation during the first boot from the local disk. This performs a configuration for all remaining unconfigured packages as if pressing `RETURN` to all questions the postinstall scripts would ask. This may not be elegant, but it works ! For safety, the client reboots for a second time later.

5.5 Main part of rcS

After installing the software packages, the default configuration of the software will not fit our local needs. Therefore we use *cfengine* and some shell scripts as the last part of the automatic installation, which is described in section 6.2. In lines 23 to 46 the type of the script is determined and it is executed. The subroutine *save_log* stores all log files on the local disk to *$FAI_LOGDIR* and to the user *$FAI_USER* on the server. Finally, we alter the boot method, in order to hinder installation again. This is done by changing the link in `/tftpboot` on the server, so the client boots another kernel, which mounts its root filesystem not from the server, but from the local disk. If the client was booted from floppy, it has to be ejected before booting. Currently we use different links in `/tftpboot` to change the kernel being booted. The following code shows the main part of `rcS`:

rcS

```
1   fai_init

    ( # execute in a subshell to get all output
    fai_setup
5   define_classes

    # partition local harddisks
    setup_harddisks.pl > /tmp/format.log 2>&1
    . /tmp/disk_var.sh
10
    # mount debian packages and install baseX_Y.tgz
    mount_packages.sh

    echo installing software may take a while
15  install_packages.pl > /tmp/software.log  2>&1

    # execute scripts; cfengine and shell scripts are known
    echo executing scripts
    cd /fai/scripts
20  for class in $classes ; do
    if [ -x $class ] && [ -f $class ]; then

       filetype=‘file $class‘
       type=
25     echo $filetype | grep -q "cfengine script" && type=cfengine
       echo $filetype | grep -q "shell script" && type=shell
       echo executing script: $class
       case $type in
```

```
        shell)
          [ -n "$verbose" ] && echo "executing shell: $class"
          echo "=====   shell: $class   =====" >> /tmp/shell.log 2>&1
          ./$class >> /tmp/shell.log 2>&1
        ;;

        cfengine)
          [ -n "$verbose" ] && echo "executing cfengine: $class"
          echo "=====   cfengine: $class   =====" >> /tmp/cfengine.log 2>&1
          ./$class --no-lock -v -f $class -D${cfclass} >> /tmp/cfengine.log 2>&1
        ;;

        *) echo "WARNING: unknown file type for file $filetype" ;;
      esac
    fi
    done

    chroot $FAI_ROOT hwclock --systohc
    date
    echo "installation completed."
    rm -f /tmp/FAI_INSTALLATION_IN_PROGRESS
    )  2>&1 | tee /tmp/rcS.log


    if [ -f /tmp/FAI_INSTALLATION_IN_PROGRESS ] ; then
        echo Error while executing commands in subshell.
        echo /tmp/FAI_INSTALLATION_IN_PROGRESS was not removed.
        echo Please look at log files for errors.
        sh
    fi

    save_log

    # now change boot device (local disk or network)
    [ -n "$FAI_USER" ] &&
        rsh -l $FAI_USER ${SERVER} "cd /tftpboot/ ; rm -f $HOSTNAME;\
                                    ln -s clusterimage $HOSTNAME"

    if [ ! -f /tmp/REBOOT ] ;then
        echo "Press <RETURN> to reboot or ctrl-c to execute a shell"
        read
    fi

    echo "rebooting now"
    cd /
    sync
    umount -a
    exec /sbin/reboot -dfi
```

The installation time is mainly determined by the amount of software that is installed on the local disk. An installation of a dataless client needing less than 50 MB data requires about two minutes using a 10 Mbit network card. An installation of a server with 310 MB of software and the same hardware needs about eight minutes. Using option `-c` in the disk configuration for a 3.5 GB partition extents the installation time by about seven minutes because it checks for bad blocks.

6 The configuration

Most files for the automatic installation process are stored in the directory tree displayed below. Only bootptab and NIS information are located in other locations but copies exists in the subdirectory `etc`.

```
lichtenstein# tree -d /files/install/fai/
/files/install/fai/
|-- class
|-- disk_config
|-- doc
|-- etc
|-- fai_scripts
|-- files
|   |-- boot
|   |   |-- System.map
|   |   |-- config
|   |   `-- vmlinuz
|   |-- etc
|   |   |-- X11
|   |   |   |-- XF86Config
|   |   |   `-- Xserver
|   |   |-- alternatives
|   |   |-- hosts
|   |   |-- hosts.allow
|   |   |-- hosts.deny
|   |   |-- hosts.equiv
|   |   |-- kbd
|   |   |   `-- default.map.gz
|   |   |-- modutils
|   |   |-- nsswitch.conf
|   |   |-- printcap
|   |   `-- rc2.d
|   |-- modules
|   |-- root
|   `-- tftpboot
|-- kernel
|-- package_config
`-- scripts
```

6.1 Scripts for defining classes

The idea of using classes in general and using certain files matching a class name for a configuration is adopted from the installation scripts by Casper Dik [16] for Solaris™. This technique proved to be very useful for our SUN workstations, so we also used it for the fully automatic installation of Linux. One simple and very efficient feature of Casper's scripts is to call a command with all files, whose file names are also a class. The following loop may implement this function in a shell script:

```
for class in $classes
do
if [ -r $config_dir/$class ]; then
   <command> $config_dir/$class
   # exit, if only the first matching file is needed
fi
done
```

A variation would be to call the command only for the first file that matches a class name. Therefore it is possible to add a new file to the configuration without changing the script. This is because the loop automaticly detects new configurations files that should be used. Unfortunately cfengine does not support this nice feature, so all classes being used in cfengine need also to be specified inside the cfengine scripts. Classes are very important for the fully automatic installation. If a client belongs to class A, we say the class A is defined. A class has no value, it is just defined or undefined. Within scripts, the variable *$classes* holds a space separated list with the names of all defined classes. Classes determine how the installation is performed. For example, an install client is configured to become a FTP server by default. If on the other hand it belongs to the class `NOFTPD`, the cfengine script disables this feature in `inetd.conf`.

Mostly a configuration is created by only changing or appending the classes to which a client belongs, making the installation of a new client very easy. Thus no additional information needs to be added to the configuration files if the existing classes suffice your needs. There are different possibilities to define classes:

1. The name of the hostname is defined to be a class.
2. Classes may be defined within a file.
3. Classes may be defined by scripts.

The last option is a very nice feature, since these scripts will define classes automatically. For example, several classes are defined only if certain hardware is identified. We use Perl [7] and shell scripts to define classes. All names of classes, except the hostname, are written in uppercase. They must not contain a hyphen, a hash or a dot, but may contain underscores. The scripts and files in `/fai/class` used to define classes are listed:

S00hostname.sh : Adds the class with the hostname, which is the first class. Additionally adds all classes that are stored in a file named as the client and the class ALL.

S01alias.sh : For all clients named roy01 to roy16, use the classes from file `roy.classes`.

S02memory.pl : Different classes are defined for different sizes of RAM. No yet used, for demonstration purpose only.

S03scsi.sh : If a SCSI device is attached, it adds the class *SCSI*. Not yet used.

S05network_card.pl : Depending on certain network cards, a class for this card is defined. These classes are used to install different loadable kernel drivers.

S07disk.pl : Defines classes depending on number of disks, their size or the overall disksize. Theses classes determine the disk layout.

S24nis.sh : If a NIS domain is defined in `/etc/bootptab`, the class *NIS* and a class with the uppercase name of the NIS domain are added. Dots are replaced by underscores.

S88dataless.sh : Add class *DATALESS* for all hosts with prefix *testclient* except *testclient99*. This script is not used, but for demonstration purpose.

S90scratch.sh : If the disk layout defines a partition `/scratch` or `/files/scratch`, the classes *NFS_SERVER* and *SCRATCH* respectively *FILES_SCRATCH* are added. This script may use classes that are defined in S07disk.pl.

S90tmp-partition.sh : If a separate partition for directory `/tmp` exists, it adds the class *TMP_PARTITION*.

S99rootpw.source : Does not add a class, but defines the variable *rootpw*. The root password is mandatory.

S99var.source : Defines some variables for cfengine.

roy.classes : A file containing classes for all clients with prefix *roy*. This file will be used by the script S01alias.sh.

faiserver : This file contains classes that are only used by client *faiserver*. S00hostname.sh will use this file.

For example, the script *S05network_card.pl* defines the classes `3C905B` and `100MBIT` for *roy01*. The first is used in cfengine to add a file in `/etc/modutils`, the latter class is not used yet, it is only added for demonstration purpose. Client *roy01* also uses the file `roy.classes` to define classes. It contains a list of classes which are defined for all clients whose hostname matches *roy??* (done by *S01alias.sh*). Using all these scripts, the client *roy01* belongs to these classes:

```
roy01 ALL DATALESS BASE NETWORK BOOT LAST REBOOT NOPCMCIA NOPPP NOTFPD
NOTELNETD NOFTPD ROY XNTP MINI_SOFT REMOTE_PRINTER HOME_CLIENT NET_9
K2_2_10 USR_LOCAL_MOUNT BIG_MEMORY 3C90X 4GB NIS INFORMATIK4711_YP
NFS_SERVER SCRATCH TMP_PARTITION
```

The defined classes are stored in the log file `FAI_CLASSES`. Hostnames should rarely be used for the configuration files in `/fai/disk_config`, `/fai/package_config` or `/fai/scripts` and subdirectories. Instead, a class is used and this class is added to the host.

Files that end in *.source* do not define classes, but may define variables for scripts that are called later. Any system administrator may write new scripts in Perl. A fundamental knowledge of Perl is not necessary[9]. There are predefined subroutines in `fai.pl`, which help writing small scripts, with a very simple syntax. To prove the correctness of a new Perl script, apply:

```
lichtenstein[~]> perl -wc S55new_script.pl
S55new_script.pl syntax OK
```

Warnings about variables, used only once do not matter. Below is an example:

S07disk.pl

```
#! /usr/bin/perl

# define classes for different disk configurations
# global variables:
# $numdisks             # number of disks
# %disksize {$device}   # size for each devie
# $sum_disk_size        # sum of all disksizes

require "fai.pl";
read_disk_info();

# rules for classes
#-------------------------------------------------------
# two SCSI disks 2-5 GB
($numdisks == 2) and
    disksize(sda,2000,5000) and
    disksize(sdb,2000,5000) and
    class("SD_2_5GB");

# one disk 1-4 GB
($numdisks == 1) and
    testsize($sum_disk_size,1000,4000) and
    class("4GB");
#-------------------------------------------------------
# do not edit beyond this line
exit;
# - - - - - - - - - - - - - - - - - - - - - - - - - - - -
sub read_disk_info {
    open ( DISK,"sfdisk -s|");
    while (<DISK>) {
    if (m!^/dev/(.+):\s+(\d+)!) {
        my ($device,$size) = ($1,$2);
        $numdisks++;
        push @devicelist,$device;
        $size /= 2048;# blocks -> Mbytes
```

[9]Learning Perl is never wasted time.

```
            $sum_disk_size += $size;
            $disksize{$device} = $size;
        }
        }
40      close DISK;
    }

    sub disksize {

45      my ($disk,$lower,$upper) = @_;
        testsize($disksize{$disk},$lower,$upper);
    }
```

Only between lines 13 and 24 changes or additions are allowed. The other parts of the script should remain unchanged. The two subroutines *class* and *classes* both print out the names of classes. The first subroutine exits the script, while the second remains to allow further checking of conditions.

6.2 Cfengine and classes

We call *cfengine*, which make the changes to the installed operating system. This is where the system is customized to our personal requirements. It is usually performed manually by the system administrator after a successful installation. For example:

- disable ftp daemon,
- set root password,
- configure DNS lookups,
- set up NIS,
- edit /etc/fstab,
- call lilo for an other kernel,
- disable unused modules (eg. pcmcia), and
- set up E-mail.

All these changes are made automatically, if they are defined in the configuration of cfengine. *Cfengine* is called for all cfengine scripts in `/fai/scripts`, that match the name of a defined class. We are also using some shell scripts, but cfengine is more appropiate for this work.

The last part of the installation is mostly done by cfengine [12]. It is a tool to set up and maintain operating systems easily. It has a rich set of commands to alter the configuration. At present we only use it during the installation, but not for maintaining the running system, although this is possible. Within cfengine, classes can be defined using modules, but we did not use this feature, because all classes which could be defined from this module before calling the module itself would have had to be declared. This is not very smart. We need a mechanism to define classes without declaring them. We therefore pass all defined classes to cfengine via the flag `-D`. Currently the following cfengine scripts are used:

```
BASE, BOOT, LAST, NETWORK, NIS, NONIS, TFTP_SERVER, X11, ALL, LAST
```

We tried different types of partitioning a configuration into several files, and the choice ranged between one long configuration file to many shorter files. A solution somewhere in the middle is probably the best choice. Often cfengine copies a "master file" from a source location to a destination. The root of the source location is `/fai/files`. The tree structure of the normal filesystem is being preserved. So if we have a master file for `/etc/nsswitch.conf`, its location is `/fai/files/etc/nsswitch.conf`. However in our configuration we have two versions of `nsswitch.conf`, one for the class *NIS* and another for class *NONIS*. If we need more than one version of a file, a directory for this file is created under the same name. So `/fai/files/etc/nsswitch.conf` converts from a file to a directory containing two files called `NIS` and `NONIS`. The part of the copy section of `NETWORK`'s cfengine configuration file looks like this:

```
copy:
   NIS::
        ${files}/etc/nsswitch.conf/NIS    dest=${target}/etc/nsswitch.conf
           m=644 o=root g=root
           force=true backup=false

   NONIS::
        ${files}/etc/nsswitch.conf/NONIS  dest=${target}/etc/nsswitch.conf
           m=644 o=root g=root
           force=true backup=false
```

Unfortunately, cfengine provides no mechanism to shorten such twin definitions. The scripts from Casper Dik can do this by automatically searching all files whose name is a class, and use the first one or all, if this make sense for the operation (not with copy).

It is advisable to document the task a class performs. Using this documentation, the creation of a configuration for a new client will become very easy because it is sufficient to choose some classes from the available classes. Here is a short description of the available classes. For more information the scripts have to be read.

BASE some base configurations

BOOT copy kernel and modules and call lilo

LAST remove old version of some files

NETWORK configure network related parts like printer, xntp, network, inetd

COMPILE select software packages for software development

KERNEL_SOFT installs kernel sources and kernel headers

KEYBOARD_GERMAN default.map for german keyboard

MINI_SOFT minimal software list

SOFT extensive software list

NIS configures system as NIS client

NONIS do not use NIS

ROY several little changes

TFTP_SERVER enable tftpd and copy clusterimage and installimage to `/tftpboot`

XNTP configures system to use NTP (Network Time Protocol)

4GB disk layout for one disk up to 4 GB

K2_2_10 kernel version 2.2.10, System.map and .config

KONGRESS1999 some special tasks for faiserver

NET_9 network related things that belongs to our class C subnet

USR_MOUNT mount `/usr` from `$bserver`

USR_LOCAL_MOUNT mount `/usr/local` from `$bserver`

USR_LOCAL_COPY make a copy of `/usr/local` to local filesystem

SCRATCH export `/scratch` to netgroup @sundomain and @linux-cluster

FILES_SCRATCH export `/files/scratch` to netgroup @sundomain and linux-cluster

FAISERVER export filesystem to netgroup @fai

NOPCMCIA remove software package pcmcia

NOPPP remove software packcage ppp

3C905B module information for the network card

NFS_SERVER select software used for a nfs server

7 Conclusions

Since FAI uses mostly scripts it is very easy to install and use. It uses only use few executables including: *cfengine, perl, sfdisk, bootpc*. Since only few changes to the root filesystem are necessary during the installation, it is very easy to set up FAI. Our installation system does not use prepared images of harddisk partitions, or save all answers to the installation questions like other tools do. It performs all steps of a normal base installation automatically using simple configuration data. Additionally, it configures the operating system to the local needs.
The FAI homepage is

`http://www.informatik.uni-koeln.de/fai`

where you can find the newest release of FAI. There is also some information on the fully automatic installation of Solaris. Please mail comments, bugs and suggestions to

`fai@informatik.uni-koeln.de`

and enjoy the fully automatic installation.

References

[1] Diskless Linux Mini Howto

[2] NFS-Root-Client Mini Howto

[3] Linux Partition Mini Howto

[4] NFS-Root Mini Howto

[5] Linux Remote-Boot Mini Howto

[6] The Linux NIS(YP)/NYS/NIS+ HOWTO

[7] Perl manuals

[8] `www.han.de/~gero/netboot/`

[9] `www.slug.org.au/etherboot/`

[10] `www.debian.org`

[11] `www.nilo.org`

[12] `www.iu.hioslo.no/cfengine`

[13] Solaris 7 Advanced Installation Guide, `docs.sun.com`

[14] `www.damtp.cam.ac.uk/linux/bootpc/`

[15] `developer.intel.com/ial/WfM/wfmspecs.htm`

[16] `ftp://ftp.fwi.uva.nl:/pub/solaris/auto-install/install.tar.gz`

[17] Sources of /usr/src/boot-floppies/utilities/ in Debian package boot-floppies

[18] Bootstrapping an Infratructure: `www.infrastructures.org/papers/bootstrap`

Appendix

create_client_root.sh

```
#! /bin/sh
# create_client_root.sh -- create installation root filesystem
# mounted readonly by all clients during installation process

installdir=/files/install/root
rcs=/files/install/fai/fai_scripts/rcS
basefile=/files/install/debian/dists/slink/main/\
disks-i386/current/base2_1.tgz
bootpc=/sbin/bootpc

echo ""
echo "create installation root filesystem in $installdir ?"
echo "type ctrl-c to abort, return to continue"

read input
if [ -d $installdir ]; then
    echo "$installdir must not exist. Please delete it."
    exit
fi

set -x
mkdir -p $installdir || exit
cd $installdir
tar zxpf $basefile
mkdir fai
rm -f etc/mtab etc/apt/sources.list
ln -s /proc/mounts etc/mtab
mv etc/init.d/rcS etc/init.d/rcS.orig

# cfengine need /var/run/ writable
rm -rf var/run
ln -s /tmp/var/run var/run

#cp $rcs etc/init.d
# make hardlinks, so you can edit the script and
# directly use the new versions
ln $rcs etc/init.d

cp $bootpc $installdir/sbin
set -
echo ""
echo do not forget to export $installdir
echo Add entry into /etc/exports and execute
echo killall -v -HUP rpc.mountd
```

install_base_root.sh

```
#! /bin/sh
# install_base_root.sh
# mount debian directory and unpack baseX_Y.tgz

basetgz=base2_1.tgz
mkdir $FAI_ROOT/debian
mount -o ro $FAI_PACKAGEDIR $FAI_ROOT/debian || exit

echo "Unpacking Debian $basetgz ..."
cd $FAI_ROOT
tar zxpf $FAI_ROOT/debian/dists/slink/main/disks-i386/current/$basetgz
```

fai.conf

```
# all (global) variables begin with FAI (fully automatic installation)
# these are global definitions for etc/init.d/rcS script

FAI_VERSION="FAI Version 1.0, Dec 1999"

# Server where to mount /usr and the Debian software packages from
FAI_NFSSERVER=$SERVER             # same as tftp server (:sa in /etc/bootptap)

# location, where log file are stored
FAI_LOGDIR=/var/log/fai

# location of master files for cfengine
FAI_FILES=/fai/files

# local disk are mounted on this directory
FAI_ROOT=/tmp/target

# FAI_USER: account on TFTP server, which saves all log-files and
# which can change the kernel that is booted via network. Configure
# .rhosts for this account, so user root can login from all install
# clients without password.  This account must have write permissions
# for /tftpboot. We are doing this with write permissions for the
# group linuxadm. chgrp linuxadm /tftpboot;chmod g+w /tftpboot
# if variable is unset, this feature is disabled
FAI_USER=fai

# full location of Debian softwarepackages
FAI_PACKAGEDIR=$FAI_NFSSERVER:/files/install/debian

export FAI_VERSION FAI_NFSSERVER FAI_LOGDIR FAI_ROOT
export FAI_USER FAI_PACKAGEDIR FAI_FILES
```