The LPIC-2 Exam Prep

Next

The LPIC-2 Exam Prep

Heinrich W. Klöpping

Beno T.J. Mesman

Piet W. Plomp

Willem A. Schreuder

Many, many Snow B.V. colleagues for peer reviewing.

Edited by

Jos Jansen

Joost Helberg

Abstract

Audience: this book is intended to help people prepare for the LPIC-2 exam. You will need to have at least 2 years of practical experience with Unix, preferably Linux. Though you may take the LPIC-2 exam without it, you should be an LPIC-1 alumnus to be allowed to the titles and rights that come with the LPIC-2 certification.

Approach: We wanted to create a set of documents that could help us and others pass the LPIC-2 exams. This book contains all the information (and more) needed to pass the exam.

Sources: Our sources of information were partly material on the Internet. Mostly practical experience of the authors and others and research done by the authors are to be credited. We try to give credit where due, but are fallible. We apologize.

Caution

While every precaution was made in the preparation of this book, we can assume no responsibility for errors or omissions. When you feel we have not given you

proper credit or feel we may have violated your rights or when you have suggestions how we may improve our work please notify us immediately so we can take corrective actions.

Organization of this book: This book is organized following the LPI Level 2 Topics, draft 4, August 11th, 2001 and written using the DocBook documentation standard.

Table of Contents

List of Figures

List of Tables

Next

Preface

Preface

Prev Next

Preface

This book reflects the efforts of a number of experienced Linux professionals to prepare for the LPIC-2 beta-exam. It is -- and always will be -- a work in progress. The authors previously obtained LPIC-1 levels, and wanted to participate in the beta-exam for various reasons: to help LPI with their beta-testing, to learn from it and to help others to pass the exam. And, last but not least: for the fun of it.

In my opinion, one of the most important mile stones set in the Linux world is the possibility for certifying our skills. Admittedly, you do not need certification to write Open Source software or Open Source documentation - your peers certainly will not ask for a certificate, but will judge you by your work.

But Linux is not just a nice toy for software-engineers. It has always been a very stable and trustworthy operating system - even more so in comparison with it's closed-source alternatives. Driven by closed source vendors' questionable license policies, security risks, bugs and vendor-lock, more and more IT-managers choose the Linux alternative. Though it's perfectly feasible to out-source system management for Linux systems - a lot of major and minor players support Linux nowadays and Linux is stable as a rock - a large number of companies prefer hiring their own sysadmins. Alas, most recruiters would not know a Linux admin if he fell on them. These recruiters need a way to certify the skills of their candidates. And the candidates need a way to prove their skills, both to themselves and to the recruiter.

A number of organizations offer Linux certification. Some of them are commercial organizations, that both certify and educate. Some of them certainly do a fine job - but I sincerely believe certification organizations should be independent, especially from educational organizations. The Linux Professional Institute fulfills these prerequisites. They also are a part of our community. Support them.

The first drafts of this book were written by a limited amount of people. We had limited time: we were offered the opportunity to do beta-exams in August 2001 and had to take the exam in September. Therefore, above all the authors had to prove to be good at cutting and pasting other peoples work. It is up to you to judge our efforts, and, hopefully, improve our work. To the many people we unintentionally forgot to give credit where due, from the bottom of our hearts: **we thank you.**

Henk Klöpping, September 2001

Prev Next
The LPIC-2 Exam Prep Home Chapter 1. Linux Kernel (2.201)

Chapter 1. Linux Kernel (2.201)

Prev Next

Chapter 1. Linux Kernel (2.201)

Revision: $Revision: 1.15 $ ($Date: 2007/01/10 19:50:39 $)

Topics

This topic has a total weight of 6 points and contains the following five objectives:

Objective 2.201.1: Kernel Components (1 point)

Candidates should be able to utilise kernel components that are necessary to specific hardware, hardware drivers, system resources and requirements. This objective includes implementing different types of kernel images, identifying stable and development kernels and patches, as well as using kernel modules.

Objective 2.201.2: Compiling a kernel (1 point)

Candidates should be able to properly configure a kernel to include or disable specific features of the Linux kernel as necessary. This objective includes compiling and recompiling the Linux kernel as needed, updating and noting changes in a new kernel, creating an initrd image and installing new kernels.

Objective 2.201.3: Patching a kernel (2 points)

Candidates should be able to properly patch a kernel to add support for new hardware. This objective also includes being able to properly remove kernel patches from already patched kernels.

Objective 2.201.4: Customizing a kernel (1 point)

Candidates should be able to customise a kernel for specific system requirements, by patching, compiling and editing configuration files as required. This objective includes being able to assess requirements for a kernel compile as well as build and configure kernel modules.

Resources: Dean01; the **man** pages for the various commands.

Prev Next

Preface Home Kernel Components (2.201.1)

Kernel Components (2.201.1)

Candidates should be able to utilise kernel components that are necessary to specific hardware, hardware drivers, system resources and requirements. This objective includes implementing different types of kernel images, identifying stable and development kernels and patches, as well as using kernel modules.

Key files, terms and utilities include:

/usr/src/linux
/usr/src/linux/Documentation
zImage
bzImage

Key knowledge area:

Kernel documentation (2.4 and 2.6)

Different types of kernel images

A kernel can be monolithic or not. A monolithic kernel is a kernel that contains all the driver code needed to use all the hardware connected to or contained within the system you are configuring. Such a kernel does not need the assistance of modules. So why doesn't everyone just build a monolithic kernel and be done with it? The main reason is that such kernels tend to be rather large and often won't fit on a single floppy disk. Another reason is that the implementation of a new improved driver for a certain piece of hardware immediately results in the need for a new kernel. This is not the case with modules as will be explained in the section called "Using kernel modules".

Most of the time, a kernel image is compressed to save space. There are two types of compressed kerneltypes: zImage and bzImage

The difference between "zImage" and "bzImage" files lies in their different layout and loading algorithms. For zImage the allowed kernelsize is 520Kb (see: Johnson01), bzImage doesn't have this restriction. As a result the bzImage kernel now is the preferred image type.

Note

Both images use gzip compression. The "bz" in "bzImage" stands for "*big zImage*", not for "bzip"!

Identifying stable and development kernels and patches

A kernel version number consists of three parts: major, minor and the patch level, all separated by periods.

The major release is incremented when a major change in the kernel is made.

The minor release is incremented when significant changes and additions are made. Even-numbered minor releases, e.g. 2.4, 2.6, 2.8, are considered stable releases and odd-numbered releases, e.g. 2.1, 2.3 are considered to be in development and are primarily used by kernel developers.

The last part of the kernel version number is the so-called patch level. As errors in the code are corrected (and/or features are added) the patch level is incremented. You should only upgrade your kernel to a higher patch level when your current kernel has a security problem or doesn't function properly or when new functionality has been added to the kernel.

Using kernel modules

Linux kernel modules are object files (.o) produced by the C compiler but not linked into a complete executable. Kernel modules can be loaded into the kernel to add functionality when needed. Most modules are distributed with the kernel and compiled along with it. Every kernel version has its own set of modules.

Modules are stored in a directory hierarchy under /lib/modules/*kernel-version*, where *kernel-version* is the string reported by **uname -r**, such as 2.2.5-15smp. Multiple module hierarchies may be available under /lib/modules if multiple kernels are installed.

Subdirectories that contain modules of a particular type exist beneath the /lib/modules/*kernel-version* directory. This grouping is convenient for administrators, but also facilitates important functionality to the command **modprobe**.

Typical module types:

block

Modules for a few block-specific devices such as RAID controllers or IDE tape drives.

cdrom

Device driver modules for nonstandard CD-ROM drives.

fs

Drivers for filesystems such as MS-DOS (the msdos.o module).

ipv4

Includes modular kernel features having to do with IP processing, such as IP masquerading.

misc

Anything that doesn't fit into one of the other subdirectories ends up here. Note that no modules are stored at the top of this tree,

net

Network interface driver modules.

scsi

Contains driver modules for the SCSI controller.

video

Special driver modules for video adapters.

Module directories are also referred to as tags in the context of module manipulation commands.

Manipulating modules

lsmod

For each kernel module loaded, display its name, size, use count and a list of other referring modules. This command yields the same information as is available in /proc/modules. On a particular laptop, for instance, the command **/sbin/lsmod** reports:

```
Module                  Size  Used by
serial_cb               1120   1
tulip_cb               31968   2
cb_enabler              2512   4 [serial_cb tulip_cb]
ds                     6384   2 [cb_enabler]
i82365                 22384   2
pcmcia_core            50080   0 [cb_enabler ds i82365]
```

The format is name, size, use count, list of referring modules. If the module controls its own unloading via a "can_unload" routine, then the user count displayed by lsmod is always -1, irrespective of the real use count.

insmod

Insert a module into the running kernel. The module is located automatically and inserted. You must be logged in as superuser to insert modules.

Frequently used options:

-s

Write results to syslog instead of the terminal.

-v

Set verbose mode.

Example 1.1. Example

The kernel was compiled with modular support for a specific sound card. To verify that this specific module, in this case maestro3.o exists, look for the file maestro3.o in the /lib/modules/*kernel-version* directory:

```
# insmod maestro3
Using /lib/modules/2.4.9/kernel/drivers/sound/maestro3.o
/lib/modules/2.4.9/kernel/drivers/sound/maestro3.o: \
   unresolved symbol ac97_probe_codec_R84601c2b
# echo $?
1
```

This **insmod** *maestro3* command yields an unresolved symbol and an exit status of 1. This is the same sort of message that might be seen when attempting to link a program that referenced variables or functions unavailable to the linker. In the context of a module insertion, such messages indicate that the functions are not available in the kernel. From the name of the missing symbol, you can see that the ac97_codec module is required to support the maestro3 module, so it is inserted first:

```
# insmod ac97_codec
Using /lib/modules/2.4.9/kernel/drivers/sound/ac97_codec.o
# insmod maestro3
Using /lib/modules/2.4.9/kernel/drivers/sound/maestro3.o
# lsmod
Module                  Size  Used by
maestro3               24272   0  (unused)
ac97_codec              8896   0  [maestro3]
```

```
serial_cb           1456   1
tulip_cb           32160   2
cb_enabler          2752   4 [serial_cb tulip_cb]
ds                 6864   2 [cb_enabler]
i82365             22512   2
pcmcia_core         49024   0 [cb_enabler ds i82365]
```

As the output from the **lsmod** command above shows, the two modules maestro3 and ac97_codec have both been loaded.

rmmod

Unless a module is in use or referred to by another module, the module is removed from the running kernel. You must be logged in as the superuser to remove modules.

Example 1.2. Example

```
# rmmod ac97_codec
ac97_codec: Device or resource busy
# echo $?
1
```

The module can't be unloaded because it is in use, in this case by the maestro3 module. The solution is to remove the maestro3 module first:

```
# rmmod maestro3
# rmmod ac97_codec
# echo $?
0
```

Issue the command **lsmod** to verify that the modules have indeed been removed from the running kernel.

modprobe

Like **insmod**, **modprobe** is used to insert modules. However, **modprobe** has the ability to load single modules, modules and their prerequisites, or all modules stored in a specific directory. The **modprobe** command can also remove modules when combined with the **-r** option.

A module is inserted with optional **symbol=value** parameters such as **irq=5** or **dma=3** .

This can be done on the command line or by specifying options in the module configuration file as will be explained later on. If the module is dependent upon other modules, they will be loaded first. The **modprobe** command determines prerequisite relationships between modules by reading modules.dep at the top of the module directory hierarchy, for instance /lib/modules/2.4.9/modules.dep.

You must be logged in as the superuser to insert modules.

Frequently used options

-a

Load all modules. When used with the -t tag, "all" is restricted to modules in the tag directory. This action probes hardware by successive module-insertion attempts for a single type of hardware, such as a network adapter. This may be necessary, for example, to probe for more than one kind of network interface.

-c

Display a complete module configuration, including defaults and directives found in /etc/modules.conf (or /etc/conf.modules, depending on your version of module utilities. Note that the LPIC objective requires you to know both names). The -c option is not used with any other options.

-l

List modules. When used with the -t tag, list only modules in directory tag.

-r

Remove module, similar to **rmmod**. Multiple modules may be specified.

-s

Display results on **syslog** instead of the terminal.

-t tag

Attempt to load multiple modules found in the directory tag until a module succeeds or all modules in tag are exhausted. This action probes hardware by successive module-insertion attempts for a single type of hardware, such as a network adapter.

-v

Set verbose mode.

Example 1.3. Example

Loading sound modules using **modprobe**.

```
# modprobe maestro3
# echo $?
0
# lsmod
Module                  Size  Used by
maestro3               24272   0  (unused)
ac97_codec              8896   0  [maestro3]
serial_cb               1456   1
tulip_cb               32160   2
cb_enabler              2752   4  [serial_cb tulip_cb]
ds                     6864   2  [cb_enabler]
i82365                 22512   2
pcmcia_core            49024   0  [cb_enabler ds i82365]
```

Both the modules have been loaded. To remove both the modules, use **modprobe -r maestro3**.

modinfo

Display information about a module from its module-object-file. Some modules contain no information at all, some have a short one-line description, others have a fairly descriptive message.

Options from the man-page:

-a, --author

Display the module's author.

-d, --description

Display the module's description.

-n, --filename

Display the module's filename.

-fformat_string, --format format_string

Let's the user specify an arbitrary format string which can extract values from the ELF section in module_file which contains the module information. Replacements consist of a percent sign followed by a tag name in curly braces. A tag name of %{filename} is always supported, even if the module has no modinfo section.

-p, --parameters

Display the typed parameters that a module may support.

-h, --help

Display a small usage screen.

-V, --version

Display the version of modinfo.

Example 1.4. Example

What information can be retrieved from the maestro3 module:

```
pug:~# modinfo maestro3
filename:    /lib/modules/2.4.9/kernel/drivers/sound/maestro3.o
description: "ESS Maestro3/Allegro Driver"
author:      "Zach Brown <zab@zabbo.net>"
parm:        debug int
parm:        external_amp int
```

Configuring modules

You may sometimes need to control the assignments of the resources a module uses, such as hardware interrupts or Direct Memory Access (DMA) channels. Other situations may dictate special procedures to prepare for, or clean up after, module insertion or removal. This type of special control of modules is configured in the file /etc/modules.conf.

Commonly used directives in /etc/modules.conf**:**

keep

The keep directive, when found before any path directives, causes the default paths to be retained and added to any paths specified.

depfile=*full_path*

This directive overrides the default location for the modules dependency file, modules.dep which will be described in the next section.

path=*path*

This directive specifies an additional directory to search for modules.

options *modulename module-specific-options*

Options for modules can be specified using the options configuration line in modules.

conf or on the **modprobe** command line. The command line overrides configurations in the file. *modulename* is the name of a single module file without the .o extension. *module-specific-options* are specified as *name=value* pairs, where the name is understood by the module and is reported using **modinfo -p**.

alias *aliasname result*

Aliases can be used to associate a generic name with a specific module, for example:

```
alias /dev/ppp         ppp_generic
alias char-major-108   ppp_generic
alias tty-ldisc-3      ppp_async
alias tty-ldisc-14     ppp_synctty
alias ppp-compress-21  bsd_comp
alias ppp-compress-24  ppp_deflate
alias ppp-compress-26  ppp_deflate
```

pre-install *module command*

This directive causes a specified shell command to be executed prior to the insertion of a module. For example, PCMCIA services need to be started prior to installing the pcmcia_core module:

```
pre-install pcmcia_core /etc/init.d/pcmcia start
```

install *module command*

This directive allows a specific shell command to override the default module-insertion command.

post-install *module command*

This directive causes a specified shell command to be executed after insertion of the module.

pre-remove *module command*

This directive causes a specified shell command to be executed prior to removal of module.

remove *module command*

This directive allows a specific shell command to override the default module-removal command.

post-remove *module command*

This directive causes a specified shell command to be executed after removal of module.

For more detailed information concerning the module-configuration file see **man modules.conf**.

Blank lines and lines beginning with a # are ignored in modules.conf.

Module Dependency File

The command **modprobe** can determine module dependencies and install prerequisite modules automatically. To do this, **modprobe** scans the first column of /lib/modules/*kernel-version*/modules.dep to find the module it is to install. Lines in modules.dep are in the following form:

module_name.o: dependency1 dependency2 ...

example for ac97_codec and maestro3:

/lib/modules/2.4.9/kernel/drivers/sound/ac97_codec.o:

/lib/modules/2.4.9/kernel/drivers/sound/maestro3.o: \
  /lib/modules/2.4.9/kernel/drivers/sound/ac97_codec.o

Here the ac97_codec module depends on no other modules and the maestro3 module depends on the ac97_codec module.

All of the modules available on the system are listed in the modules.dep file and are referred to with their full path and filenames, including their .o extension. Those that are not dependent on other modules are listed, but without dependencies. Dependencies that are listed are inserted into the kernel by **modprobe** first, and after they have been successfully inserted, the subject module itself can be loaded.

The modules.dep file must be kept current to ensure the correct operation of **modprobe**. If module dependencies were to change without a corresponding modification to modules.dep, then **modprobe** would fail because a dependency would be missed. As a result, modules.dep needs to be (re)created each time the system is booted. Most distributions will do this by executing **depmod -a** automatically when the system is booted.

This procedure is also necessary after any change in module dependencies. The **depmod** command is actually a link to the same executable as **modprobe**. The functionality of the command differs depending on which name is used to execute it.

Prev | Up | Next
Chapter 1. Linux Kernel (2.201) | Home | Compiling a Kernel (2.201.2)

Compiling a Kernel (2.201.2)

Candidates should be able to properly configure a kernel to include or disable specific features of the Linux kernel as necessary. This objective includes compiling and recompiling the Linux kernel as needed, updating and noting changes in a new kernel, creating an initrd image and installing new kernels.

Key files, terms and utilities include:

make *config, xconfig, menuconfig, oldconfig, mrproper, zImage, bzImage, modules, modules_install*
mkinitrd (both Red hat and Debian based)
/usr/src/linux
/etc/lilo.conf
/boot/grub/menu.lst or /boot/grub/grub.conf

Getting the kernel sources

The kernel sources for the latest Linux kernel can be found at *The Linux Kernel Archives* .

The generic filename is always in the form linux-*kernel-version*.tar.gz or linux-*kernel-version*.tar.bz2. For example, linux-2.4.18.tar.gz is the kernel archive for version "2.4.18".

A common location to store and unpack kernel sources is /usr/src.

If there isn't enough free space on /usr/src to unpack the sources it is possible to unpack the source in anoher directory. Creating a softlink after unpacking from /usr/src/linux to the linux subdirectory in that location ensures easy access to the source.

The source code will be available as a compressed tar archive. Compression is done using **gzip** (.gz extention) or **bzip2** (.bz2 extention). Decompressing and unpacking can be done using **gunzip** or **bunzip2** followed by unpacking the resulting archive with **tar**, or directly with **tar** using the *z* (.gz) or *j* (.bz2) options:

```
# gunzip linux-2.4.18.tar.gz
# tar xf linux-2.4.18.tar
```

or

```
# tar xjf linux-2.4.18.tar.bz2
```

See manpages for **tar**, **gzip**, **bzip2** for more information.

Creating a .config file

The first step in compiling a kernel is setting up the kernel configuration which is saved in the .config file. There are more than 500 options for the kernel, such as filesystem, SCSI and networking support. Many of the options list kernel features that can be either compiled directly into the kernel or compiled as modules.

Some selections imply a group of other selections. For example, when you indicate that you wish to include SCSI support, additional options become available for specific SCSI drivers and features.

The results of all of these choices are stored in the kernel configuration file /usr/src/linux/.config, which is a plain text file that lists the options as shell variables.

To begin, set the current working directory to the top of the source tree:

```
# cd /usr/src/linux
```

There are several ways to set up .config. Although you can do so, you should not edit the file manually. Instead, select one of the three interactive approaches. An additional option is available to construct a default configuration. Each set up is started using **make**.

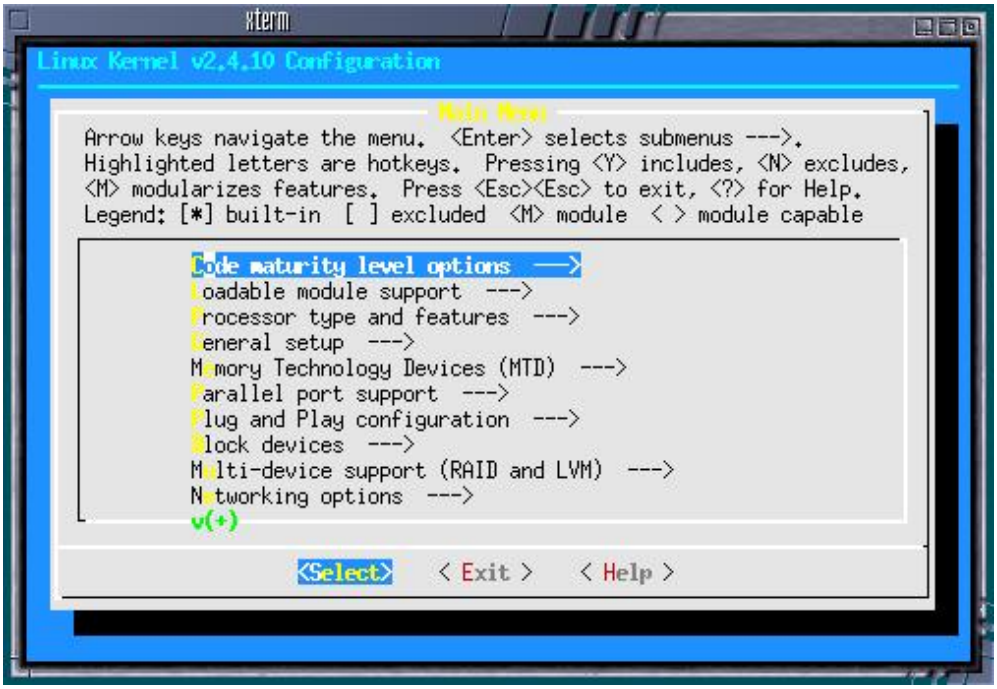
make config

Running **make config** is the most rudimentary of the automated kernel-configuration methods and does not depend on full-screen display capabilities of your terminal. The system sequentially presents you with questions concerning kernel options. This method can, admittedly, get a bit tedious and has the disadvantage that you must answer all the questions before being able to save your .config file and exit. However, it is helpful if you do not have sufficient capability to use one of the menu-based methods. A big drawback is that you can't move back and forward through the various questions. An example session looks like this:

```
# make config
rm -f include/asm
( cd include ; ln -sf asm-i386 asm)
/bin/sh scripts/Configure arch/i386/config.in
#
# Using defaults found in arch/i386/defconfig
#
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers \
   (CONFIG_EXPERIMENTAL) [N/y/?] y
*
* Loadable module support
*
Enable loadable module support (CONFIG_MODULES) [Y/n/?]
```

make menuconfig

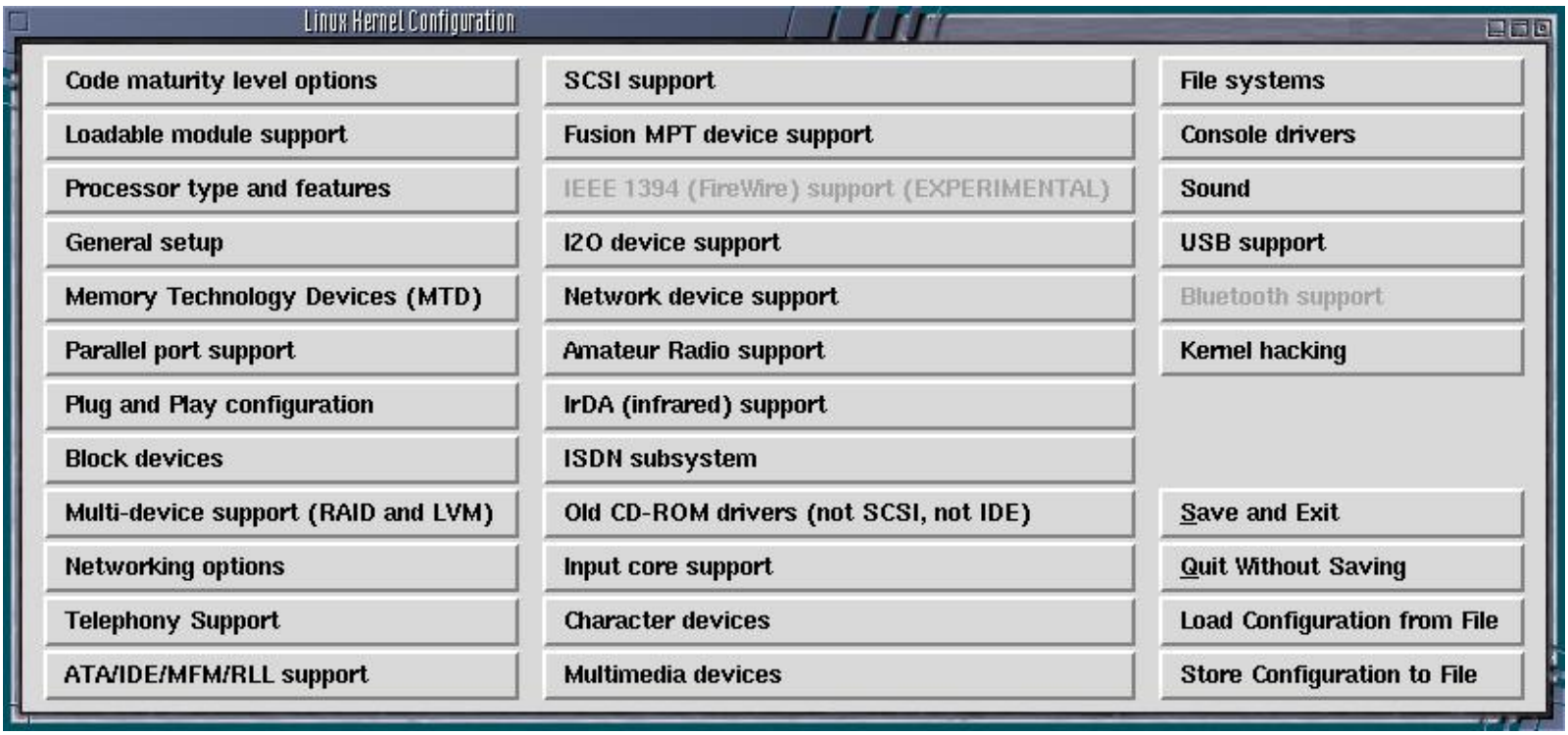
This configuration method is more intuitive and can be used as an alternative to **make config**. It creates a text-mode-windowed environment where you may use arrow and other keys to configure the kernel. The **make menuconfig** command is illustrated below in an xterm.



The **make menuconfig** menu display.

make xconfig

If you are running the X Window System, the **make xconfig** command presents a GUI menu with radio buttons to make the selections. The figure below illustrates the top-level **make xconfig** window.



The **make xconfig** top-level window.

make oldconfig

The **make oldconfig** command creates a new .config file, using the old .config and options found in the source, only requiring user interaction for options that that were previously not configured (new options), for instance after addition of new functionality, of upgrading to a later kernel release.

When using the .config from a previous kernel release, first copy the .config from the previous kernel-version to the /usr/src/linux/ directory and then run **make oldconfig**. The .config will be moved to .config.old and a new .config is created. You will only be prompted for the answers to new questions; the questions for which no answer can be found in the .config.old file.

> **Note**
>
> Be sure to make a backup of .config before upgrading the kernel source, because the distribution might contain a default .config file, overwriting your old file.

> **Note**
>
> make xconfig and make menuconfig runs will start with reconstructing the .config file, using the current .config and (new default) options found in the source code. As a result .config will contain at least all the previously set and the new default options after writing the new .config file and exiting the session without any manual changes.

Compiling the kernel

The following sequence of make commands leads to the building of the kernel and to the building and installation of the modules.

1. make dep
2. make clean
3. make zImage/bzImage
4. make modules
5. make modules_install

make dep

The *dep* object examins source files for dependencies. The resulting table of dependencies is stored in a file called .depend. There will be a .depend file in each directory containing source files. The .depend files are automatically included in subsequent **make** operations.

make clean

The "clean" object removes old output files that may exist from previous kernel builds. These include core files, system map files and others.

make zImage/bzImage

The **zImage** and **bzImage** objects both effectively build the kernel. The difference between these two is explained in another paragraph.

After compilation the kernel image can be found in the /usr/src/linux/arch/i386/boot directory (on i386 systems).

make modules

The **modules** object builds the modules: device drivers and other items that were configured as modules.

make modules_install

The **modules_install** object installs all previously built modules under /lib/modules/*kernel-version* The *kernel-version* directory will be created if non-exsistent.

Installing the new kernel

After the new kernel has been compiled, the system can be configured to boot it.

The first step is to put a copy of the new bzImage on the boot partition. The name of the kernel file should preferably contain the kernel-version number, for example: vmlinuz-2.4.10:

```
# cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz-2.4.10
```

The currently available kernel versions can be found in the directory /boot/ as shown below:

```
# ls -l /boot
-rw-r--r--   1 root     root       426824 Mar 23  2001 vmlinuz-2.2.18pre21
-rw-r--r--   1 root     root       739881 Sep 27 10:42 vmlinuz-2.4.10
-rw-r--r--   1 root     root       728156 Sep 21 15:24 vmlinuz-2.4.9
```

Next, configure the bootmanager to contain the new kernel.

Lilo and version-specific names

lilo.conf could look like this:

```
# Boot up this one by default.
default=vmlinuz-2410

image=/boot/vmlinuz-2.4.10
      label=linux-2410
      read-only
image=/boot/vmlinuz-2.4.9
      label=linux-249
      read-only
image=/boot/vmlinuz-2.2.18pre21
      label=linux-2218
      read-only
```

Run **lilo** to add the new kernel and reboot to activate the new kernel.

Lilo and fixed names

Another way to enable the new kernel is by creating symbolic links in the root directory to the kernel images in /boot. It is a good practice to have at least two kernels available: the kernel that has been used so far (and which is working properly) and the newly compiled one (which might not work since it hasn't been tested yet). One is called vmlinuz and the other is called vmlinuz.old. Like this the labels in /etc/lilo.conf don't need to be changed after installation of a new kernel.

First point vmlinuz.old to the latest known-working kernel image (in our example: /boot/vmlinuz-2.4.9) and point vmlinuz to the new kernel image (in our example: /boot/vmlinuz-2.4.10):

```
# ls -l vmlinuz*
lrwxrwxrwx 1 root root 19 Sep 19 17:33 vmlinuz -> /boot/vmlinuz-2.4.9
lrwxrwxrwx 1 root root 25 Sep 19 17:33 vmlinuz.old -> /boot/vmlinuz-2.2.18pre21
# rm /vmlinuz.old
# ln -s /boot/vmlinuz-2.4.9 /vmlinuz.old
# rm /vmlinuz
# ln -s /boot/vmlinuz-2.4.10 /vmlinuz
```

lilo.conf could look like this:

```
# Boot up this one by default.
```

```
default=Linux

image=/vmlinuz
      label=Linux
      read-only
image=/vmlinuz.old
      label=LinuxOLD
      read-only
```

Run **lilo** to add the new kernel and reboot to activate.

```
# lilo
Added Linux *
Added LinuxOLD
```

If the new kernel doesn't work, reboot again, press SHIFT to have the LILO boot loader activate the boot menu and select the last known working kernel to boot.

> **Note**
>
> Don't forget to run lilo after changing lilo.conf, otherwise the new kernel can't be booted because lilo.conf is not consulted at boot time.

Grub bootmanager

Just like with Lilo both version specific and fixed kernel image names kan be used with Grub.

A kernel configuration entry in /boot/grub/menu.lst (or /boot/grub/grub.conf) could look something like:

```
title         GNU/Linux, kernel 2.6.8
root          (hd0,0)
kernel        /boot/vmlinuz-2.6.8 root=/dev/hda1 ro
initrd        /boot/initrd.img-2.6.8
savedefault
boot
```

mkinitrd

The list of key files, terms and utilities also mentions **mkinitrd**. The following text can be found on *The Linux Head Quarters* :

> Using the initial RAM disk (initrd)
>
> initrd adds the capability to load a RAM disk by the boot loader. This RAM disk can then be mounted as the root filesystem and programs can be run from it. Afterwards, a new root filesystem can be mounted from a different device. The previous root (from initrd) is then either moved to the directory /initrd or it is unmounted.
>
> initrd is mainly designed to allow system startup to occur in two phases, where the kernel comes up with a minimum set of compiled-in drivers, and where additional modules are loaded from initrd.
>
> Operation
>
> When using initrd, the system boots as follows:
>
> 1. the boot loader loads the kernel and the initial RAM disk
> 2. the kernel converts initrd into a "normal" RAM disk and frees the memory used by initrd.

3. initrd is mounted read-write as root
4. **linuxrc** is executed (this can be any valid executable, including shell scripts; it is run with uid 0 and can do basically everything **init** can do)
5. when **linuxrc** terminates, the "real" root filesystem is mounted
6. if a directory /initrd exists, the initrd is moved there, otherwise, initrd is unmounted
7. the usual boot sequence (e.g. invocation of **/sbin/init**) is performed on the root filesystem

Note that moving initrd from / to /initrd does not involve unmounting it. It is therefore possible to leave processes running on initrd (or leave filesystems mounted, but see below) during that procedure. However, if /initrd doesn't exist, initrd can only be unmounted if it is not used by anything. If it can't be unmounted, it will stay in memory.

Also note that filesystems mounted under initrd continue to be accessible, but their /proc/mounts entries are not updated. Also, if /initrd doesn't exist, initrd can't be unmounted and will "disappear" and take those filesystems with it, thereby preventing them from being re-mounted. It is therefore strongly suggested to generally unmount all filesystems (except of course for the root filesystem, but including /proc) before switching from initrd to the "normal" root filesystem.

In order to de-allocate the memory used for the initial RAM disk, you have to execute **freeramdisk** after unmounting /initrd.

initrd adds the following new options to the boot command line options:

initrd= (LOADLIN only)

Loads the specified file as the initial RAM disk. When using LILO, you have to specify the RAM disk image file in /etc/lilo.conf, using the INITRD configuration variable.

noinitrd

initrd data is preserved but it is not converted to a RAM disk and the "normal" root filesystem is mounted. initrd data can be read from /dev/initrd. Note that the data in initrd can have any structure in this case and doesn't necessarily have to be a filesystem image. This option is used mainly for debugging.

Note that /dev/initrd is read-only and that it can only be used once. As soon as the last process has closed it, all data is freed and /dev/initrd can't be opened any longer.

root=/dev/ram

initrd is mounted as root, and **/linuxrc** is started. If no **/linuxrc** exists, the normal boot procedure is followed, with the RAM disk still mounted as root. This option is mainly useful when booting from a floppy disk. Compared to directly mounting an on-disk filesystem, the intermediate step of going via initrd adds a little speed advantage and it allows the use of a compressed file system. Also, together with LOADLIN you may load the RAM disk directly from CDrom or disk, hence having a floppy-less boot from CD, e.g.: **E:\loadlin E:\bzImage root=/dev/ram initrd=E:\rdimage**

Installation

First, the "normal" root filesystem has to be prepared as follows:

```
# mknod /dev/initrd b 0 250
# chmod 400 /dev/initrd
# mkdir /initrd
```

If the root filesystem is created during the boot procedure (i.e. if you're creating an install floppy), the root filesystem creation procedure should perform these operations.

Note that neither /dev/initrd nor /initrd are strictly required for correct operation of initrd, but it is a lot easier to experiment with initrd if you have them, and you may also want to use /initrd to pass data to the "real" system.

Second, the kernel has to be compiled with RAM disk support and with support for the initial RAM disk enabled. Also, all components needed to execute programs from initrd (e.g. executable format and filesystem) must be compiled into the kernel).

Third, you have to create the RAM disk image. This is done by creating a filesystem on a block device and then by copying files to it as needed. With recent kernels, at least three types of devices are suitable for that:

a floppy disk (works everywhere but it's painfully slow)
a RAM disk (fast, but allocates physical memory)
a loopback device (the most elegant solution)

We'll describe the RAM disk method:

1. make sure you have a RAM disk device /dev/ram (block, major 1, minor 0)
2. create an empty filesystem of the appropriate size, e.g.

```
# mke2fs -m0 /dev/ram 300
(if space is critical, you may want to use the Minix FS instead of Ext2)
```

3. mount the filesystem on an appropriate directory, e.g.

```
# mount -t ext2 /dev/ram /mnt
```

4. create the console device:

```
# mkdir /mnt/dev
# mknod /mnt/dev/tty1 c 4 1
```

5. copy all the files that are needed to properly use the initrd environment. Don't forget the most important file, **/linuxrc** Note that **/linuxrc** must be given execute permission.
6. unmount the RAM disk

```
# umount /dev/ram
```

7. copy the image to a file

```
# dd if=/dev/ram bs=1k count=300 of=/boot/initrd
```

8. deallocate the RAM disk

```
# freeramdisk /dev/ram
```

For experimenting with initrd, you may want to take a rescue floppy (e.g. rescue.gz from Slackware) and only add a symbolic link from **/linuxrc** to **/bin/sh**, e.g.

```
# gunzip /dev/ram
# mount -t minix /dev/ram /mnt
# ln -s /bin/sh /mnt/linuxrc
# umount /dev/ram
# dd if=/dev/ram bs=1k count=1440 of=/boot/initrd
# freeramdisk /dev/ram
```

Finally, you have to boot the kernel and load initrd . With LOADLIN, you simply execute

LOADLIN initrd=
e.g. LOADLIN C:\LINUX\VMLINUZ initrd=C:\LINUX\INITRD

With LILO, you add the option INITRD= to either the global section or to the section of the respective kernel in /etc/lilo.conf, e.g.

image = /vmlinuz
initrd = /boot/initrd

and run **/sbin/lilo**

Now you can boot and enjoy using initrd.

Setting the root device

By default, the standard settings in the kernel are used for the root device, i.e. the default compiled in or set with **rdev** , or what was passed with root=xxx on the command line, or, with LILO, what was specified in /etc/lilo.conf It is also possible to use initrd with an NFS-mounted root; you have to use the nfs_root_name and nfs_root_addrs boot options for this.

It is also possible to change the root device from within the initrd environment. In order to do so, /proc has to be mounted. Then, the following files are available:

/proc/sys/kernel/real-root-dev
/proc/sys/kernel/nfs-root-name
/proc/sys/kernel/nfs-root-addrs

real-root-dev can be changed by writing the number of the new root FS device to it, e.g.

echo 0x301 >/proc/sys/kernel/real-root-dev

for /dev/hda1. When using an NFS-mounted root, nfs-root-name and nfs-root-addrs have to be set accordingly and then real-root-dev has to be set to 0xff, e.g.

```
# echo /var/nfsroot >/proc/sys/kernel/nfs-root-name
# echo 193.8.232.2:193.8.232.7::255.255.255.0:idefix \
  >/proc/sys/kernel/nfs-root-addrs
# echo 255 >/proc/sys/kernel/real-root-dev
```

If the root device is set to the RAM disk, the root filesystem is not moved to /initrd, but the boot procedure is simply continued by starting init on the initial RAM disk.

Usage scenarios

The main motivation for implementing initrd was to allow for modular kernel configuration at system installation. The procedure would work as follows:

1. systems boots from floppy or other media with a minimal kernel (e.g. support for RAM disks, initrd, a.out, and the ext2 FS) and loads initrd
2. **/linuxrc** determines what is needed to (1) mount the "real" root FS (i.e. device type, device drivers, filesystem) and (2) the distribution media (e.g. CD-ROM, network, tape, ...). This can be done by asking the user, by auto-probing, or by using a hybrid approach.
3. **/linuxrc** loads the necessary modules
4. **/linuxrc** creates and populates the root file system (this doesn't have to be a very usable system yet)

5. **/linuxrc** unmounts the root filesystem and possibly any other filesystems it has mounted, sets /proc/sys/kernel/..., and terminates
6. the root filesystem is mounted
7. now that we're sure that the filesystem is accessible and intact, the boot loader can be installed
8. the boot loader is configured to load an initrd with the set of modules that was used to bring up the system (e.g. /initrd can be modified, then unmounted, and finally, the image is written from /dev/ram to a file)
9. now the system is bootable and additional installation tasks can be performed

The key role of initrd here is to re-use the configuration data during normal system operation without requiring the use of a bloated "generic" kernel or re-compilation or re-linking of the kernel.

A second scenario is for installations where Linux runs on systems with different hardware configurations in a single administrative domain. In such cases, it is desirable to generate only a small set of kernels (ideally only one) and to keep the system-specific part of configuration information as small as possible. In this case, a common initrd could be generated with all the necessary modules. Then, only **/linuxrc** or a file read by it would have to be different.

A third scenario might result in more convenient recovery disks, because information like the location of the root FS partition doesn't have to be provided at boot time, and the system loaded from initrd can use a user-friendly dialog and can also perform some sanity checks (or even some form of auto-detection).

Last but not least, CDrom distributors may use it for better installation from CD, either using a LILO boot floppy and bootstrapping a bigger ramdisk via initrd from CD, or using LOADLIN to directly load the ramdisk from CD without need of floppies.

Since initrd is a fairly generic mechanism, it is likely that additional uses will be found.

Patching a Kernel (2.201.3)

Candidates should be able to properly patch a kernel to add support for new hardware. This objective also includes being able to properly remove kernel patches from already patched kernels.

Key files, terms and utilities include:

patch
gzip
bzip2
Makefile

Patching a kernel

A kernel patchfile contains a difference listing produced by the **diff** command. The **patch** command is used to apply the contents of the patchfile to the kernel sources.

Patching a kernel is very straightforward:

1. Place patchfile in /usr/src.
2. Change directory to /usr/src.
3. Uncompress the patchfile.
4. Use **patch** to apply the patchfile to the kernel source: **patch -p1 <** patchfile
5. Check for failures.
6. Build the kernel.

If **patch** is unable to apply a part of a patch, it puts that part in a reject file. The name of a reject file is the name of the output file plus a .rej suffix, or a # if the addition of ".rej" would generate a filename that is too long. In case even the addition of a mere # would result in a filename that is too long, the last character of the filename is replaced with a #.

The patch command and it's most common options:

-p*number* , --strip=*number*

Strip the smallest prefix containing *number* leading slashes from each file name found in the patch file. A sequence of one or more adjacent slashes is counted as a single

slash. This controls how file names found in the patch file are treated, in case you keep your files in a different directory than the person who sent out the patch. For example, supposing the file name in the patch file was /u/howard/src/blurfl/blurfl.c, then setting -p0 gives the entire file name unmodified, and setting -p1 gives u/howard/src/blurfl/blurfl.c.

-s , --silent , --quiet

Work silently, unless an error occurs.

-E , --remove-empty-files

Remove output files that are empty after the patches have been applied. Normally this option is unnecessary, since patch can examine the time stamps on the header to determine whether a file should exist after patching. However, if the input is not a context diff or if patch is conforming to POSIX, patch does not remove empty patched files unless this option is given. When patch removes a file, it also attempts to remove any empty ancestor directories.

-R , --reverse

Assume that this patch was created with the old and new files swapped. **patch** attempts to swap each hunk around before applying it and rejects will come out in the swapped format. The -R option does not work with **ed** diff scripts because there is too little information to reconstruct the reverse operation. If the first hunk of a **patch** fails, **patch** reverses the hunk to see if it can be applied that way. If it can, you are asked if you want to have the -R option set. If it can't, the patch continues to be applied normally. (Note: this method cannot detect a reversed patch if it is a normal diff and if the first command is an append (i.e. it should have been a delete) since appends always succeed, due to the fact that a null context matches anywhere. Luckily, most patches add or change lines rather than delete them, so most reversed normal diffs begin with a delete, which fails, triggering the heuristic.)

For more information consult the man-pages of the **diff** command and the **patch** command.

Removing a kernel patch from a production kernel

A kernel patch can be removed from a production kernel by removing it from the production kernel source tree and compiling a new kernel.

To remove the patch from the production kernel simply try to apply the patch a second time, patch will ask you if it should assume "-R" as the following example illustrates:

```
# patch -p1 < patch-2.4.13-pre3
patching file linux/Documentation/Configure.help
Reversed (or previously applied) patch detected!  Assume -R? [n] y
```

Prev | Up | Next
Compiling a Kernel (2.201.2) | Home | Customizing a Kernel (2.201.4)

Customizing a Kernel (2.201.4)

Candidates should be able to customise a kernel for specific system requirements, by patching, compiling and editing configuration files as required. This objective includes being able to assess requirements for a kernel compile as well as build and configure kernel modules.

Key files, terms, and utilities include:

patch
make
/usr/src/linux
/proc/sys/kernel/
/etc/conf.modules, /etc/modules.conf
insmod, lsmod, modprobe
kmod
kerneld

Most of the key files, terms and utilities mentioned above have been explained earlier in this chapter, the ones that haven't been mentioned will be mentioned in the paragraphs that follow.

/proc/sys/kernel/ is a directory in the /proc pseudo filesystem which contains the following entries:

```
$ ls -l /proc/sys/kernel/
-rw-------   1 root    root        0 Oct 22 14:11 cad_pid
-rw-------   1 root    root        0 Oct 22 14:11 cap-bound
-rw-r--r--   1 root    root        0 Oct 22 14:11 core_uses_pid
-rw-r--r--   1 root    root        0 Oct 22 14:11 ctrl-alt-del
-rw-r--r--   1 root    root        0 Oct 22 14:11 domainname
-rw-r--r--   1 root    root        0 Oct 22 14:11 hostname
-rw-r--r--   1 root    root        0 Oct 22 14:11 hotplug
-rw-r--r--   1 root    root        0 Oct 22 14:11 modprobe
-rw-r--r--   1 root    root        0 Oct 22 14:11 msgmax
-rw-r--r--   1 root    root        0 Oct 22 14:11 msgmnb
-rw-r--r--   1 root    root        0 Oct 22 14:11 msgmni
```

```
-r--r--r--   1 root    root         0 Oct 22 14:11 osrelease
-r--r--r--   1 root    root         0 Oct 22 14:11 ostype
-rw-r--r--   1 root    root         0 Oct 22 14:11 overflowgid
-rw-r--r--   1 root    root         0 Oct 22 14:11 overflowuid
-rw-r--r--   1 root    root         0 Oct 22 14:11 panic
-rw-r--r--   1 root    root         0 Oct 22 14:11 printk
dr-xr-xr-x   2 root    root         0 Oct 22 14:11 random
-rw-r--r--   1 root    root         0 Oct 22 14:11 rtsig-max
-r--r--r--   1 root    root         0 Oct 22 14:11 rtsig-nr
-rw-r--r--   1 root    root         0 Oct 22 14:11 sem
-rw-r--r--   1 root    root         0 Oct 22 14:11 shmall
-rw-r--r--   1 root    root         0 Oct 22 14:11 shmmax
-rw-r--r--   1 root    root         0 Oct 22 14:11 shmmni
-rw-r--r--   1 root    root         0 Oct 22 14:11 tainted
-rw-r--r--   1 root    root         0 Oct 22 14:11 threads-max
-r--r--r--   1 root    root         0 Oct 22 14:11 version
```

Some files can be used to get information, for instance to show the version of the running kernel:

```
cat /proc/sys/kernel/osrelease
```

Some files can also be used to *set* information in the kernel. For instance, the following will tell the kernel not to loop on a panic, but to auto-reboot after 20 seconds:

```
echo 20 >  /proc/sys/kernel/panic
```

kmod versus kerneld

What are they?

Both **kmod** and **kerneld** make dynamic loading of kernel-modules possible. A module is loaded when the kernel needs it. (Modules are explained earlier is this chapter).

What is the difference between them?

kerneld is a daemon, **kmod** is a thread in the kernel itself.

The communication with **kerneld** is done through System V IPC. **kmod** operates directly from the kernel and does not use System V IPC thereby making it an optional module.

kmod replaces **kerneld** as of Linux kernel 2.2.x.

What do they have in common?

Both facilitate dynamic loading of kernel modules.

Both use **modprobe** to manage dependencies and dynamic loading of modules.

Manual loading of modules with **modprobe** or **insmod** is possible without the need for **kmod** or **kerneld**.

In both cases, the kernel-option CONFIG_MODULES must be set to enable the usage of modules.

Enabling kmod

To enable the use of **kmod**, a kernel must be compiled with the kernel-option CONFIG_KMOD enabled. This can only be done if the kernel-option CONFIG_MODULES has also been enabled.

Chapter 2. System Startup (2.202)

Prev Next

Chapter 2. System Startup (2.202)

Revision: $Revision: 1.13 $ ($Date: 2004/08/13 08:14:21 $)

This topic has a total weight of 5 points and contains the following 2 objectives:

Objective 2.202.1 Customizing system startup and boot processes (2 points)

> This topic includes being able to edit the appropriate system startup scripts to customize standard system run levels and boot processes, interacting with run levels and creating custom initrd images as needed.

Objective 2.202.2 System recovery (3 points)

> This topic includes being able to properly configure and navigate the standard Linux filesystem, configuring and mounting various filesystem types, and manipulating filesystems to adjust for disk space requirements or device additions.

Customizing system startup and boot processes (2.202.1)

This topic includes being able to edit the appropriate system startup scripts to customize standard system run levels and boot processes, interacting with run levels and creating custom initrd images as needed.

Key files, terms and utilities include:

/etc/init.d/
/etc/inittab
/etc/rc.d
mkinitrd Described in the section called “mkinitrd”

The Linux Boot process

A description of the boot process can also be found at *Vanderbilt University*.

The Linux boot process can be logically divided into six parts. They are as follows:

1. Kernel loader loading, setup, and execution (bootsect.s)

   In this step the file bootsect.s is loaded into memory by the BIOS. bootsect.s then sets up a few parameters and loads the rest of the kernel into memory.
2. Parameter setup and switch to 32-bit mode (boot.s)

   After the kernel has been loaded, boot.s takes over. It sets up a temporary IDT and GDT (explained later on) and handles the switch to 32-bit mode.

   Detailed information on IDT, GDT and LDT can be found on *sandpile.org - The world's leading source for pure technical x86 processor information.*
3. Kernel decompression (compressed/head.s)

   The kernel is stored in a compressed format. This head.s (since there is another head.s) decompresses the kernel.
4. Kernel setup (head.s)

   After the kernel is decompressed, head.s (the second one) takes over. The real GDT and IDT are created, as is a basic memory-paging table.
5. Kernel and memory initialization (main.c)

   This step is the most complex. The kernel now has control and sets up all remaining parameters and initializes everything remaining. Virtual memory is setup completely and the first processes are created.
6. Init process creation (main.c)

   In the final step of booting, the Init process is created.

Kernel Loader (linux/arch/i386/boot/bootsect.s)

When the computer is first turned on, BIOS loads the boot sector of the boot disk into memory at location 0x7C00. This first sector corresponds to the bootsect.s file. The BIOS will only copy 512 bytes, so the kernel loader must be small. The code that is loaded by the BIOS must be able to load the remaining portions of the operating system and pass control onto the next file.

The first thing that bootsect.s does when it is loaded is to move itself to the memory location 0x9000. This is to avoid any possible conflicts in memory. The code then jumps to the new copy located at 0x9000. After this, an area in memory is set aside (0x4000-12) for a new disk parameter table. To make it so that more than one sector can be read from the disk at a time, we will try to find the largest number of sectors that can be read at a time. This will help speed reads from the disk when we begin loading the rest of the kernel.

Before this is done, setup.s is loaded into memory in the memory space above bootsect.s, 0x9020. This allows setup.s to be jumped to after the kernel has been loaded. Now the disk parameter table is created. Basically, the code tries to read 36 sectors, if that fails it tries 18, 15, then if all else fails it uses 9 as the default.

If at any point there is an error, little can be done. In most cases, bootsect.s will just keep trying to do what it was doing when the error occurred. Usually this will end in an unbroken loop that can only be resolved by rebooting by hand.

At last we are ready to copy the kernel into memory. bootsect.s goes into a loop that reads the first 508Kb from the disk and places it into memory starting at 0x10000. After the kernel is loaded into RAM, bootsect.s jumps to 0x9020, where setup.s is loaded.

Parameter Setup (linux/arch/i386/boot/setup.s)

setup.s makes sure that all hardware information has been collected and gathers more information if necessary. It first verifies that it is loaded at 0x9020. After this is verified, setup.s does the following:

1. Gets main memory size
2. Sets keyboard repeat rate to the maximum
3. Retrieves video card information
4. Collects monitor information for the terminal to use
5. Gets information about the first and possibly second hard drive using BIOS
6. Looks to see if there is a mouse (a pointing device) attached to the system

All of the information that setup.s collects is stored for later use by device drivers and other areas of the system. Like bootsect.s, if an error occurs little can be done. Most errors are "handled" by an infinite loop that has to be reset manually.

The next step in the booting process needs to use virtual memory. This can only be used on a x86 by switching from real mode to protected mode. After all information has been gathered by setup.s, it does a few more housekeeping chores to get ready for the switch to 32-bit mode.

First, all interrupts are disabled. Once the system is in 32-bit mode, no more BIOS calls can be made. The area of memory at 0x1000 is where the BIOS handlers were loaded when the system came up. We no longer need these, so to get the compressed kernel out of the way, setup.s moves the kernel from 0x10000 to 0x1000. This provides room for a temporary IDT (Interrupt Descriptor Table) and GDT (Global Descriptor Table). The GDT is only setup to have the system in memory. All paging is disabled, so that described memory locations correspond to actual memory addresses. At this point, extended (or high) memory is enabled.

Setup also resets any present coprocessor and reconfigures the 8259 Programmable

Interrupt Controller. All that remains now is for the protected bit mask to be set, and the processor is in 32-bit mode. After the switch has been made, setup.s lets processing continue at /compressed/head.s to uncompress the kernel.

Kernel Decompression (linux/arch/i386/boot/compressed/head.s)

This first head.s uncompresses the kernel into memory. The kernel is gzip-compressed to make sure that it can fit into the 508Kb that bootsect.s will load. When the kernel is compiled, bootsect.s, head.s , and /compressed/head.s are not compressed and are appended to the front of the compressed kernel. They are the only three files that must remain uncompressed.

head.s decompresses the kernel to address 0x1000000. This corresponds to the 1Mb boundary in memory. head.s does a bit of error checking before it decompresses the kernel to ensure that there is enough memory available in high memory.

Right before the decompression is done, the flags register is reset and the area in memory where setup.s was is cleared. This is to put the system in a better known state. After the decompression, control is passed to the now decompressed head.s.

Kernel Setup (linux/arch/i386/kernel/head.s)

The second head.s is responsible for setting up the permanent IDT and GDT, as well as a basic paging table. Before anything is done, the flags register is again reset. The first page in the paging system is setup at 0x5000. This page is filled by the information gathered in setup.s by copying it from its location at 0x9000.

Next the processor type is determined. For 586s (Pentium) and higher there is a processor command that returns the type of processor. Unfortunately, the 386 and 486 do not have this feature so some tricks have to be employed. Each processor has only certain flags, so by trying to read and write to them you can determine the type of processor. If a coprocessor is present that is also detected.

After that, the IDT and GDT are set up. The table for the IDT is set up. Each interrupt gets an 8-byte descriptor. Each descriptor is initially set to ignore_int. This means that nothing will happen when the interrupt is called. All that ignore_int does is, is save the registers, print "unknown interrupts", and then restore the registers.

Each IDT descriptor is divided into four two-byte sections. The top four bytes are called the WW, while the bottom four are the CW. The WW contains a two-byte offset, a P-flag set to 1, and a Descriptor Privilege Level. The CW has a selector and an offset. In total the IDT can contain up to 256 entries.

At this point the code sets up memory paging. In the x86 architecture, virtual memory uses three descriptors to establish an address: a Page Directory, a Page Table, and a Page Frame. The Page Directory is a table of all of the pages and what processes they match to.

The Page Directory contains an index into the Page Table. The Page Table maps the virtual address to the beginning of a physical page in memory. The Page Frame and an offset use the beginning address of the physical page and can retrieve an actual location in memory. The three structures are setup by head.s. They make it so that the first 4Mb of memory is in the Page Directory. The kernel's virtual address is set to 0xC0000000, or the top of the last gigabyte of memory.

Each memory address in an x86 has three parts. The first is the index into the Page Directory. The result of this index is the start of a specific Page Table. The second part of the 32-bit address is an offset into the Page Table. The Page Table has a 32-bit entry that corresponds to that offset. The top 20 bits are used to get an actual physical address. The lower 12 bits are used for administrative purposes. The physical address corresponds to the start of a physical page. The third part of the 32-bit address is an offset within this page, equal to a real memory location.

Almost everything is set up at this point. Now control is passed to the main function in the kernel. Main.c gains control.

Kernel Initialization (linux/init/main.c)

All remaining setup and initialization functions are called from main.c. Paging, the IDT and most of the other systems have been initialized by now. Main.c will make sure everything is in its proper place before it tries to start some processes and give control to init.c.

A call to the function start_kernel() is made. In essence all that start_kernel() does is run through a list of init functions that needed to be called. Such things as paging, traps, IRQs, process schedules and more are setup. The important work has already been done for memory and interrupts. Now all that has to be done is to have all of the tables filled in.

Init process creation (linux/init/main.c)

After all of the init functions have been called main.c tries to start the init process. main.c tries three different copies of **init** in order. If the first doesn't work, it tries the second, if that one doesn't work it goes to the third. Here are the file names for the three init versions:

1. /etc/init
2. /bin/init
3. /sbin/init

If none of these three **init**s work, then the system goes into single user mode. **init** is needed to log in multiple users and to manage many other tasks. If it fails, then the single user mode creates a shell and the system goes from there.

What happens next, what does /sbin/init **do?**

init is the parent of all processes, it reads the file /etc/inittab and creates processes based on its contents. One of the things it usually does is **spawn gettys** so that users can log in. It also defines so called "runlevels".

A "runlevel" is a software configuration of the system which allows only a selected group of processes to exist.

init can be in one of the following eight runlevels

runlevel 0 (reserved)

Runlevel 0 is used to halt the system.

runlevel 1 (reserved)

Runlevel 1 is used to get the system in single user mode.

runlevel 2-5

Runlevels 2,3,4 and 5 are multi-user runlevels.

runlevel 6

Runlevel 6 is used to reboot the system.

runlevel 7-9

Runlevels 7, 8 and 9 are also valid. Most of the Unix variants don't use these runlevels. On a particular Debian Linux System for instance, the /etc/rc<runlevel>.d directories, which we'll discuss later, are not implemented for these runlevels, but they could be.

runlevel s or S

Runlevels s and S are internally the same runlevel S which brings the system in "single-user mode". The scripts in the /etc/rcS.d directory are executed when booting the system. Although runlevel S is not meant to be activated by the user, it can be.

runlevels A, B and C

Runlevels A, B and C are so called "on demand" runlevels. If the current runlevel is "2" for instance, and an **init A** command is executed, the things to do for runlevel "A" are done but the actual runlevel remains "2".

Configuring /etc/inittab

As mentioned earlier, **init** reads the file /etc/inittab to determine what it should do. An entry in this file has the following format:

id:runlevels:action:process

Included below is an example /etc/inittab **file.**

```
# The default runlevel.
id:2:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fall through in case of emergency.
z6:6:respawn:/sbin/sulogin

# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
```

Description of an entry in /etc/inittab**:**

id

The id-field uniquely identifies an entry in the file /etc/inittab and can be 1-4 characters in length. For gettys and other login processes however, the id field should contain the suffix of the corresponding tty, otherwise the login accounting might not work.

runlevels

This field contains the runlevels for which the specified action should be taken.

action

The "action" field can have one of the following values:

respawn

The process will be restarted whenever it terminates, (e.g. getty).

wait

The process will be started once when the specified runlevel is entered and **init** will wait for its termination.

once

The process will be executed once when the specified runlevel is entered.

boot

The process will be executed during system boot. The runlevels field is ignored.

bootwait

The process will be executed during system boot, while **init** waits for its termination (e.g. /etc/rc). The runlevels field is ignored.

off

This does absolutely nothing.

ondemand

A process marked with an on demand runlevel will be executed whenever the specified ondemand runlevel is called. However, no runlevel change will occur (on demand runlevels are "a", "b", and "c").

initdefault

An initdefault entry specifies the runlevel which should be entered after system boot. If none exists, init will ask for a runlevel on the console. The process field is ignored. In the example above, the system will go to runlevel 2 after boot.

sysinit

The process will be executed during system boot. It will be executed before any boot or bootwait entries. The runlevels field is ignored.

powerwait

The process will be executed when the power goes down. **init** is usually informed about this by a process talking to a UPS connected to the computer. **init** will wait for the process to finish before continuing.

powerfail

As for powerwait, except that **init** does not wait for the process's completion.

powerokwait

This process will be executed as soon as **init** is informed that the power has been restored.

powerfailnow

This process will be executed when **init** is told that the battery of the external UPS is almost empty and the power is failing (provided that the external UPS and the monitoring process are able to detect this condition).

ctrlaltdel

The process will be executed when **init** receives the SIGINT signal. This means that someone on the system console has pressed the CTRL-ALT-DEL key combination. Typically one wants to execute some sort of shutdown either to get into single-user level or to reboot the machine.

kbdrequest

The process will be executed when init receives a signal from the keyboard handler that a special key combination was pressed on the console keyboard. Basically you want to map some keyboard combination to the "KeyboardSignal" action. For example, to map Alt-Uparrow for this purpose use the following in your keymaps file: alt keycode 103 = KeyboardSignal.

process

This field specifies the process that should be executed. If the process field starts with a "+", **init** will not do utmp and wtmp accounting. Some gettys insist on doing their own housekeeping. This is also a historic bug.

The /etc/init.d/rc script

For each of the runlevels 0-6 there is an entry in /etc/inittab that executes **/etc/init.d/rc ?** where "?" is 0-6, as you can see in following line from the earlier example above:

```
l2:2:wait:/etc/init.d/rc 2
```

So, what actually happens is that **/etc/init.d/rc** is called with the runlevel as a parameter.

The directory /etc contains several, runlevel specific, directories which in their turn contain runlevel specific symbolic links to scripts in /etc/init.d/. Those directories are:

```
$ ls -d /etc/rc*
/etc/rc.boot  /etc/rc1.d  /etc/rc3.d  /etc/rc5.d  /etc/rcS.d
/etc/rc0.d    /etc/rc2.d  /etc/rc4.d  /etc/rc6.d
```

As you can see, there also is a /etc/rc.boot directory. This directory is obsolete and has been replaced by the directory /etc/rcS.d. At boot time, the directory /etc/rcS.d is scanned first and then, for backwards compatibility, the /etc/rc.boot.

The name of the symbolic link either starts with an "S" or with a "K". Let's examine the /etc/rc2.d directory:

```
$ ls /etc/rc2.d
K20gpm      S11pcmcia   S20logoutd  S20ssh      S89cron
S10ipchains  S12kerneld  S20lpd      S20xfs      S91apache
S10sysklogd  S14ppp      S20makedev  S22ntpdate  S99gdm
S11klogd     S20inetd    S20mysql    S89atd      S99rmnologin
```

If the name of the symbolic link starts with a "K", the script is called with "stop" as a parameter to stop the process. This is the case for K20gpm, so the command becomes **K20gpm stop**. Let's find out what program or script is called:

```
$ ls -l /etc/rc2.d/K20gpm
lrwxrwxrwx 1 root root 13 Mar 23 2001 /etc/rc2.d/K20gpm -> ../init.d/gpm
```

So, **K20gpm stop** results in **/etc/init.d/gpm stop**. Let's see what happens with the "stop" parameter by examining part of the script:

```
#!/bin/sh
#
# Start Mouse event server
...
case "$1" in
  start)
```

```
    gpm_start
    ;;
  stop)
    gpm_stop
    ;;
  force-reload|restart)
    gpm_stop
    sleep 3
    gpm_start
    ;;
  *)
    echo "Usage: /etc/init.d/gpm {start|stop|restart|force-reload}"
    exit 1
esac
```

In the case..esac the first parameter, $1, is examined and in case its value is "stop", **gpm_stop** is executed.

On the other hand, if the name of the symbolic link starts with an "S", the script is called with "start" as a parameter to start the process.

The scripts are executed in a lexical sort order of the filenames.

Let's say we've got a daemon **SomeDaemon**, an accompanying script /etc/init.d/SDscript and we want **SomeDaemon** to be running when the system is in runlevel 2 but not when the system is in runlevel 3.

As you've read earlier, this means that we need a symbolic link, starting with an "S", for runlevel 2 and a symbolic link, starting with a "K", for runlevel 3. We've also determined that the daemon **SomeDaemon** is to be started after **S19someotherdaemon** which implicates S20 and K80 since starting/stopping is symmetrical, i.e. that what is started first is stopped last. This is accomplished with the following set of commands:

```
# cd /etc/rc2.d
# ln -s ../init.d/SDscript S20SomeDaemon
# cd /etc/rc3.d
# ln -s ../init.d/SDscript K80SomeDaemon
```

Should you wish to manually start, restart or stop a process, it is good practice to use the appropriate script in /etc/init.d/ , e.g. **/etc/init.d/gpm restart** to initiate the restart of the process.

Prev | Next

Customizing a Kernel (2.201.4) | Home | System recovery (2.202.2)

System recovery (2.202.2)

This topic includes being able to properly configure and navigate the standard Linux filesystem, configuring and mounting various filesystem types, and manipulating filesystems to adjust for disk space requirements or device additions.

Key files, terms and utilities include:

LILO
init
inittab
mount
fsck

Influencing the regular boot process

The regular boot process is the process that normally takes place when (re)booting the system. This process can be influenced by entering something at the LILO prompt. What can be influenced will be discussed in the following sections, but first we must activate the prompt. The LILO prompt is activated if LILO sees that one of the **Shift**, **Ctrl** or **Alt** keys is pressed, or **CapsLock** or **ScrollLock** is set after LILO is loaded.

Choosing another kernel

If you've just compiled a new kernel and you're experiencing difficulties with the new kernel, chances are that you'd like to revert to the old kernel. Of course you've kept the old kernel and added a label to lilo.conf which will show up if you press **Tab** or **?** on the LILO prompt. Choose the old kernel, then solve the problems with the new one.

Booting into single user mode or a specific runlevel

This can be useful if, for instance, you've installed a graphical environment which isn't functioning properly. You either don't see anything at all or the system doesn't reach a finite state because is keeps trying to start X over and over again.

Booting into single user mode or into another runlevel where the graphical environment isn't running will give you access to the system so you can correct the problem. To boot into single user mode type the name of the label corresponding to the kernel you'd like started

followed by an "s", "S" or the word "single". If the label is "Linux", you can type one of the following after the LILO prompt:

```
LILO: Linux s
LILO: Linux S
LILO: Linux single
```

If you have defined a runlevel, let's say runlevel 4, which is a text-only runlevel, you can type the following line to boot into runlevel 4 and regain access to your system:

```
LILO: Linux 4
```

Passing parameters to the kernel

If a device doesn't work

A possible cause can be that the device driver in the kernel has to be told to use another irq and/or another I/O port. BTW: This is only applicable if support for the device has been compiled into the kernel, not if you're using a loadable module.

As an example, let's pretend we've got a system with two identical ethernet-cards for which support is compiled into the kernel. By default only one card will be detected, so we need to tell the driver in the kernel to probe for both cards. Suppose the first card is to become eth0 with an address of 0x300 and an irq of 5 and the second card is to become eth1 with an irq of 11 and an address of 0x340. This is done at the LILO bootprompt as shown below:

```
LILO: Linux ether=5,0x300,eth0 ether=11,0x340,eth1
```

Be careful only to include white space between the two "ether=" parameters.

If you've lost the root password

This never happens because, being a well-trained system administrator, you've written the password down on a piece of paper, put it in an envelope and placed it in the vault.

In case you haven't done this, shame on you!

The trick is to try to get a shell with root-privileges but without having to type the root password. As soon as that's accomplished, you can remove the root password by clearing root's second field in the file /etc/passwd (the fields are separated by colons), do a reboot, login as root and use **passwd** to set the new root password.

First reboot the system and type the following at the LILO boot prompt:

LILO: Linux init=/bin/bash

This will give you the shell but it's possible that the default editor **vi** can't be found because it's located on a filesystem that is not mounted yet (which would be the case on my system because **vi** on my system is located in /usr/bin which is not mounted). It's possible to do a **mount -a** now which will mount all the filesystems mentioned in /etc/fstab, except the ones with the **noauto** keyword, but you won't see that the filesystems have been mounted when you type **mount** after that because **mount** and **umount** keep a list of mounted filesystems in the file /etc/mtab which couldn't be updated because the root (/) filesystem under which /etc/mtab resides is mounted read-only.

Keeping all this in mind, the steps to follow after we've acquired the shell are:

```
# mount -o remount,rw /   'remount / readable and writable
# mount -a                'mount all
# mount                   'show mounted filesystems
# vi /etc/passwd          'and clear the second field for root
# sync                    'write buffers to disk
# umount -a               'unmount filesystems
# mount -o remount,ro /   'remount / read-only again
<Ctrl><Alt><Del>
login: root               'login as root without password
# passwd                  'type the new password
```

Ahem, now is a good time to write the root password ...

The Rescue Boot process

When fsck is started but fails

During boot, on my Debian system, this is done by /etc/rcS.d/S30check.fs. All filesystems are checked based on the contents of /etc/fstab.

If the command **fsck** returns an exit status larger than 1, the command has failed. The exit status is the result of one or more of the following conditions:

0 - No errors
1 - File system errors corrected
2 - System should be rebooted

4 - File system errors left uncorrected
8 - Operational error
16 - Usage or syntax error
128 - Shared library error

If the command has failed you'll get a message:

fsck failed. Please repair manually

"CONTROL-D" will exit from this shell and
continue system startup.

If you don't press **Ctrl-D** but enter the root password, you'll get a shell, in fact **/sbin/sulogin** is launched, and you should be able to run **fsck** and fix the problem if the root filesystem is mounted read-only.

Alternatively, as is described in the next section, you can boot from a home-made disc or from the distribution boot media.

If your root (/) filesystem is corrupt

Using a home-made bootfloppy

The default root filesystem is compiled into the kernel. This means that if a kernel has been built on a system that uses /dev/hda2 as the root filesystem and you've put the kernel on a floppy and boot from floppy, the kernel will still try to mount /dev/hda2 as the root filesystem.

The only way to circumvent this is to tell the kernel to mount another device as the root filesystem. Let's say we've cooked up a floppy with a root filesystem on it and a kernel that is an exact copy of our current kernel. All we have to do is boot from floppy and tell the kernel to use the root filesystem from the floppy. This is done at the LILO prompt:

LILO: Linux root=/dev/fd0

Using the distribution's bootmedia

A lot of distributions come with two floppy disks and one or more CD's. One of the floppy disks is called the "bootdisk" or the "rescuedisk" the other is called the "rootdisk".

You can boot from the "bootdisk" and the system will ask you to enter the "rootdisk". After

both disks have been processed, you've got a running system with the root filesystem (/) in a RAMdisk.

In case you're wondering why we didn't boot from CD; we could have, if the system supports booting from CD. Remember to set the boot-order in the BIOS to try to boot from CD-ROM first and then HDD.

As soon as we've booted from the disks or from CD we can get a shell with root-privileges but without having to type the root password by pressing **Alt-F2**. It is now possible to manually run **fsck** on the umounted filesystems.

Let's assume your root partition was /dev/hda2. You can then run a filesystem check on the root filesystem by typing **fsck -y /dev/hda2**. The "-y" flag prevents **fsck** from asking questions which you must answer (this can result in a lot of **Enter**s) and causes **fsck** to use "yes" as an answer to all questions.

Although your current root (/) filesystem is completely in RAM, you can mount a filesystem from harddisk on an existing mountpoint in RAM, such as /target or you can create a directory first and then mount a harddisk partition there.

After you've corrected the errors, don't forget to **umount** the filesystems you've mounted before you reboot the system, otherwise you'll get a message during boot that one or more filesystems have not been cleanly umounted and **fsck** will try to fix it again.

Chapter 3. Filesystem (2.203)

Prev Next

Chapter 3. Filesystem (2.203)

Revision: $Revision: 1.13 $ ($Date: 2007/01/10 19:47:17 $)

This objective has a weight of 10 points and contains the following three objectives:

Objective 2.203.1; Operating the Linux filesystem (3 points)

> The formal LPI objective states: "Candidates should be able to properly configure and navigate the standard Linux filesystem. This objective includes configuring and mounting various filesystem types. Also included is manipulating filesystems to adjust for disk space requirements or device additions."

Objective 2.203.2; Maintaining a Linux filesystem (4 points)

> The formal LPI objective states: "Candidates should be able to properly maintain a Linux filesystem using system utilities. This objective includes manipulating standard filesystems."

Objective 2.203.3; Creating and configuring filesystem options (3 points)

> The formal LPI objective states: "Candidates should be able to configure automount filesystems. This objective includes configuring automount for network and device filesystems. Also included is creating non ext2 filesystems for devices such as CD-ROMs."

Operating The Linux Filesystem (2.203.1)

The formal LPI objective states: "Candidates should be able to configure automount filesystems. This objective includes configuring automount for network and device filesystems. Also included is creating filesystems for devices such as CD-ROMs."

Key files, terms and utilities include:

The concept of the fstab configuration
Tools and utilities for handling SWAP partitions and files
/etc/fstab
mount/umount

/etc/mtab
sync
swapon/swapoff
/proc/mounts

Resources: LinuxRef07, Wirzenius98.

The File Hierarchy

Historically, the location of certain files and utilities has not always been standard (or fixed). This has led to problems with development and upgrading between different "distributions" of Linux. The Linux directory structure (or file hierarchy) was based on existing flavors of UNIX, but as it evolved, certain inconsistencies developed. These were often small things like the location (or placement) of certain configuration files, but it resulted in difficulties porting software from host to host.

To equalize these differences a file standard was developed. This is an evolving process, to date, resulting in a fairly static model for the Linux file hierarchy.

The top level of the Linux file hierarchy is referred to as the root (or /). The root directory typically contains several other directories including:

bin/	Required boot-time binaries
boot/	Boot configuration files for the OS loader and kernel image
dev/	Device files
etc/	System configuration files and scripts
home/	User/Sub branch directories
lib/	Main OS shared libraries and kernel modules
lost+found/	Storage directory for “recovered” files
mnt/	Temporary point to connect devices to
proc/	Pseudo directory structure containing information about the kernel, currently running processes and resource allocation
root/	Linux (non-standard) home directory for the root user. Alternate location being the / directory itself
sbin/	System administration binaries and tools
tmp/	Location of temporary files
usr/	Difficult to define - it contains almost everything else including local binaries, libraries, applications and packages (including X Windows)

var/	Variable data, usually machine specific. Includes spool directories for mail and news

Generally, the root should not contain any additional files - a possible exception would be mount points for various purposes.

Filesystems

A filesystem is build of the methods and data structures that a operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem. Thus, one might say "I have two filesystems" meaning one has two partitions on which one stores files, or that one might say that one is using the "XFS filesystem", meaning the type of the filesystem.

Important

The difference between a disk or partition and the filesystem it contains is important. A few programs (including, reasonably enough, programs that create filesystems) operate directly on the raw sectors of a disk or partition; if a filesystem is already there it will be destroyed or seriously corrupted. Most programs operate on a filesystem, and therefore won't work on a partition that doesn't contain one (or that contains one of the wrong type).

Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called making a filesystem.

Most UNIX filesystem types have a similar general structure, although the exact details vary quite a bit. The central concepts are superblock, inode, data block, directory block, and indirection block. The superblock contains information about the filesystem as a whole, such as its size (the exact information here depends on the filesystem). An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically. These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

Creating Filesystems

Before a partition can be mounted (or used), a filesystem must first be installed on it - with ext2, this is the process of creating i-nodes and data blocks.

This process is the equivalent of formatting the partition (similar to MSDOS's **format** command). Under Linux, the command to create a filesystem is called **mkfs**.

The command is issued in the following way:

```
mkfs  [-c][ -t fstype ]  filesystem [ blocks ]
```

e.g.

```
mkfs -t ext2 /dev/fd0   # Make a ext2 filesystem on a floppy
```

where:

-c

forces a check for bad blocks

-t fstype

specifies the filesystem type. For most filesystem types there is a shorthand for this e.g.: **mkfs -t ext2** can also be called as **mke2fs** or **mkfs.ext2** and **mkfs -t vfat** or **mkfs -t msdos** can also be called as **mkfs.vfat**, **mkfs.msdos** or **mkdosfs**

filesystem

is either the device file associated with the partition or device OR is the directory where the file system is mounted (this is used to erase the old file system and create a new one)

Be aware that creating a filesystem on a device with an existing filesystem will cause all data on the old filesystem to be erased.

Mounting and Unmounting

To attach a partition or device to the directory hierarchy you must mount its associated device file. To do this, a mount point has to be created - this is simply a directory where the device will be attached. This directory will exist on a previously mounted device (with the exception of the root directory (/) which is a special case) and will be empty. If the directory is not empty, then the files in the directory will not be visible while the device is mounted to it, but will reappear after the device has been disconnected (or unmounted).

To mount a device, use the mount command:

```
mount [switches] device_file mount_point
```

With some devices, mount will detect what type of filesystem exists on the device, however it is more usual to use mount in the form of:

```
mount [switches] -t file_system_type device_file mount_point
```

Generally, only the root user can use the mount command - mainly due to the fact that the device files are owned by root. For example, to mount the first partition on the second hard drive off the /usr directory and assuming it contained the ext2 filesystem, you'd enter the command:

```
mount -t ext2 /dev/hdb1 /usr
```

A common device that is mounted is the floppy drive. A floppy disk generally contains the FAT, also known as msdos, filesystem (but not always) and is mounted with the command:

```
mount -t msdos /dev/fd0 /mnt
```

Note that the floppy disk was mounted under the /mnt directory. This is because the /mnt directory is the usual place to temporarily mount devices.

To see what devices you currently have mounted, simply type the command **mount**. Typing it on my system reveals:

```
/dev/hda3 on / type ext2 (rw)
/dev/hda1 on /dos type msdos (rw)
none on /proc type proc (rw)
/dev/cdrom on /cdrom type iso9660 (ro)
/dev/fd0 on /mnt type msdos (rw)
```

Each line tells me what device file is mounted, where it is mounted, what filesystem type each partition is and how it is mounted (*ro* = read only, *rw* = read/write). Note the strange entry on line three - the proc filesystem? This is a special "virtual" filesystem used by Linux systems to store information about the kernel, processes and current resource usages. It is actually part of the system's memory - in other words, the kernel sets aside an area of memory in which it stores information about the system. This same area is mounted onto the filesystem so user programs have access to this information.

The information in the proc filesystem can also be used to see what filesystems are mounted

by issuing the command:

```
$ cat /proc/mounts
/dev/root / ext2 rw 0 0
proc /proc proc rw 0 0
/dev/hda1 /dos msdos rw 0 0
/dev/cdrom /cdrom iso9660 ro 0 0
/dev/fd0 /mnt msdos rw 0 0
```

To release a device and disconnect it from the filesystem, the umount command is used. It is issued in the form:

umount device_file

or

umount mount_point

For example, to release the floppy disk, you'd issue the command:

umount /mnt

or

umount /dev/fd0

Again, you must be the root user or a user with privileges to do this. You can't unmount a device/mount point that is in use by a user (the user's current working directory is within the mount point) or is in use by a process. Nor can you unmount devices/mount points which in turn have devices mounted to them. All of this raises the question - how does the system know which devices to mount when the OS boots?

In true UNIX fashion, there is a file which governs the behavior of mounting devices at boot time. In Linux, this file is /etc/fstab. So what is in the file? An example line from the fstab file uses the following format:

device_file mount_point file_system_type mount_options [n] [n]

The first three fields are self explanatory; the fourth field, *mount_options* defines how the device will be mounted (this includes information of access mode *ro*/*rw*, execute permissions and other information) - information on this can be found in the **mount** man pages (note that this field usually contains the word "defaults"). The fifth and sixth fields are used by the system utilities **dump** and **fsck** respectively - see the next section for details.

Swap

Linux can use either a normal file in the filesystem or a separate partition for swap space. A swap partition is faster, but it is easier to change the size of a swap file (there's no need to repartition the whole hard disk, and possibly install everything from scratch). When you know how much swap space you need, you should use a swap partition, but if you are uncertain, you can use a swap file first, use the system for a while so that you can get a feel for how much swap you need, and then make a swap partition when you're confident about its size. But it is recommended to use a separate partition, because this excludes chances of file system fragmentation, which would reduce performance. Also, by using a separate swap partition, it can be guaranteed that the swap region is at the fastest location of the disk. On current HDDs this is the beginning. It is possible to use several swap partitions and/or swap files at the same time. This means that if you only occasionally need an unusual amount of swap space, you can set up an extra swap file at such times, instead of keeping the whole amount allocated all the time.

The command **mkswap** is used to initialize a swap partition or a swap file. The partition or file needs to exist before it can be initialized. A swap partition is created with a disk partitioning tool like **fdisk** and a swap file can be created with:

dd if=/dev/zero of=swapfile bs=1024 count=65535

When the partition or file is created, it can be initialized with:

mkswap {device|file}

An initialized swap space is taken into use with **swapon**. This command tells the kernel that the swap space can be used. The path to the swap space is given as the argument, so to start swapping on a temporary swap file one might use the following command:

swapon /swapfile

or, when using a swap partition:

swapon /dev/hda8

Swap spaces can be used automatically by listing them in the file /etc/fstab:

```
/dev/hda8       none       swap       sw    0    0
/swapfile       none       swap       sw    0    0
```

The startup scripts will run the command **swapon -a**, which will start swapping on all the swap spaces listed in /etc/fstab. Therefore, the swapon command is usually used only when extra swap is needed. You can monitor the use of swap spaces with **free**. It will tell the total amount of swap space used:

```
$ free
      total       used       free     shared    buffers     cached
Mem:        127148     122588       4560          0       1584      69352
-/+ buffers/cache:      51652      75496
Swap:       130748      57716      73032
```

The first line of output (Mem:) shows the physical memory. The total column does not show the physical memory used by the kernel, which is loaded into the RAM memory during the boot process. The used column shows the amount of memory used (the second line does not count buffers). The free column shows completely unused memory. The shared column shows the amount of memory shared by several processes; The buffers column shows the current size of the disk buffer cache.

That last line (Swap:) shows similar information for the swap spaces. If this line is all zeroes, swap space is not activated.

The same information, in a slightly different format, can be shown by using **cat** on the file /proc/meminfo:

```
$ cat /proc/meminfo
total:    used:    free:  shared: buffers:  cached:
Mem:  130199552 125177856  5021696        0  1622016 89280512
Swap: 133885952 59101184 74784768
MemTotal:       127148 kB
MemFree:          4904 kB
MemShared:           0 kB
Buffers:          1584 kB
Cached:          69120 kB
SwapCached:      18068 kB
Active:          80240 kB
Inactive:        31080 kB
HighTotal:           0 kB
HighFree:            0 kB
```

```
LowTotal:      127148 kB
LowFree:         4904 kB
SwapTotal:     130748 kB
SwapFree:       73032 kB
```

[Prev](#) System recovery (2.202.2)

[Home](#)

[Next](#) Maintaining a Linux Filesystem (2.203.2)

Maintaining a Linux Filesystem (2.203.2)

The formal LPI objective states: "Candidates should be able to properly maintain a Linux filesystem using system utilities." This objective includes manipulating a standard ext2 filesystem.

Key files, terms and utilities include:

fsck (fsck.ext2)
badblocks
mke2fs
dumpe2fs
debuge2fs
tune2fs

Resources: Bandel97; the **man** pages for **e2fsck**, **badblocks**, **dumpe2fs**, **debugfs** and **tune2fs**.

Good disk maintenance requires periodic disk checks. Your best tool is **fsck**, and should be run at least monthly. Default checks will normally be run after 20 system reboots, but if your system stays up for weeks or months at a time, you'll want to force a check from time to time. Your best bet is performing routine system backups and checking your lost+found directories from time to time.

The frequency of the checks at system reboot can be changed with **tune2fs**. This utility can also be used to change the mount count, which will prevent the system from having to check all filesystems at the 20th reboot (which can take a long time).

The **dumpe2fs** utility will provide important information regarding hard disk operating parameters found in the superblock, and **badblocks** will perform surface checking. Finally, surgical procedures to remove areas grown bad on the disk can be accomplished using **debugfs**.

fsck (fsck.e2fs)

fsck is a utility to check and repair a Linux filesystem. In actuality **fsck** is simply a front-end for the various filesystem checkers (**fsck.***fstype*) available under Linux.

Fsck is called automatically at system startup. If the filesystem is marked "not clean", the maximum mount count is reached or the time between check is exceeded, the filesystem is checked. To change the maximum mount count or the time between checks, use **tune2fs**.

Frequently used options to **fsck** include:

-s

Serialize **fsck** operations. This is a good idea if you checking multiple filesystems and the checkers are in an interactive mode.

-A

Walk through the /etc/fstab file and try to check all filesystems in one run. This option is typically used from the /etc/rc system initialization file, instead of multiple commands for checking a single filesystem.

The root filesystem will be checked first. After that, filesystems will be checked in the order specified by the *fs_passno* (the sixth) field in the /etc/fstab file. Filesystems with a *fs_passno* value of 0 are skipped and are not checked at all. If there are multiple filesystems with the same pass number, fsck will attempt to check them in parallel, although it will avoid running multiple filesystem checks on the same physical disk.

-R

When checking all filesystems with the -A flag, skip the root filesystem (in case it's already mounted read-write).

Options which are not understood by fsck are passed to the filesystem-specific checker. These arguments must not take arguments, as there is no way for fsck to be able to properly guess which arguments take options and which don't. Options and arguments which follow the-- are treated as filesystem-specific options to be passed to the filesystem-specific checker.

The filesystem checker for the ext2 filesystem is called **fsck.e2fs** or **e2fsck**. Frequently used options include:

-a

This option does the same thing as the -p option. It is provided for backwards compatibility only; it is suggested that people use -p option whenever possible.

-c

This option causes e2fsck to run the **badblocks(8)** program to find any blocks which are bad on the filesystem, and then marks them as bad by adding them to the bad block inode.

-C

This option causes e2fsck to write completion information to the specified file descriptor so that the progress of the filesystem check can be monitored. This option is typically used by programs which are running e2fsck. If the file descriptor specified is 0, e2fsck will print a completion bar as it goes about its business. This requires that e2fsck is running on a video console or terminal.

-f

Force checking even if the filesystem seems clean.

-n

Open the filesystem read-only, and assume an answer of "no" to all questions. Allows e2fsck to be used non-interactively. (Note: if the -c, -l, or -L options are specified in addition to the -n option, then the filesystem will be opened read-write, to permit the bad-blocks list to be updated. However, no other changes will be made to the filesystem.)

-p

Automatically repair ("preen") the filesystem without any questions.

-y

Assume an answer of "yes" to all questions; allows **e2fsck** to be used non-interactively.

tune2fs

tune2fs is used to "tune" a filesystem. This is mostly used to set filesystem check options, such as the *maximum mount count* and the *time between filesystem checks*.

It is also possible to set the *mount count* to a specific value. This can be used to 'stagger' the mount counts of the different filesystems, which ensures that at reboot not all filesystems will be checked at the same time.

So for a system that contains 5 partitions and is booted approximately once a month you could do the following to stagger the mount counts:

tune2fs -c 5 -C 0 *partition1*
tune2fs -c 5 -C 1 *partition2*
tune2fs -c 5 -C 2 *partition3*
tune2fs -c 5 -C 3 *partition4*
tune2fs -c 5 -C 4 *partition5*

The maximum mount count is 20, but for a system that is not frequently rebooted a lower value is advisable.

Frequently used options include:

-c max-mount-counts

Adjust the maximum mount count between two filesystem checks. If max-mount-counts is 0 then the number of times the filesystem is mounted will be disregarded by e2fsck(8) and the kernel. Staggering the mount-counts at which filesystems are forcibly checked will avoid all filesystems being checked at one time when using journalling filesystems.

You should strongly consider the consequences of disabling mount-count-dependent checking entirely. Bad disk drives, cables, memory and kernel bugs could all corrupt a filesystem without marking the filesystem dirty or in error. If you are using journalling on your filesystem, your filesystem will never be marked dirty, so it will not normally be checked. A filesystem error detected by the kernel will still force an fsck on the next reboot, but it may already be too late to prevent data loss at that point.

-C mount-count

Set the number of times the filesystem has been mounted. Can be used in conjunction with -c to force an fsck on the filesystem at the next reboot.

-i interval-between-checks[d|m|w]

Adjust the maximal time between two filesystem checks. No postfix or d result in days, m in months, and w in weeks. A value of zero will disable the time-dependent checking.

It is strongly recommended that either -c (mount-count-dependent) or -i (time-dependent) checking be enabled to force periodic full e2fsck(8) checking of the filesystem. Failure to do so may lead to filesystem corruption due to bad disks, cables or memory or kernel bugs to go unnoticed, until they cause data loss or corruption.

-m reserved-blocks-percentage

Set the percentage of reserved filesystem blocks.

-r reserved-blocks-count

Set the number of reserved filesystem blocks.

dumpe2fs

dumpe2fs prints the super block and blocks group information for the filesystem present on device.

-b

print the blocks which are reserved as bad in the filesystem.

-h

only display the superblock information and not any of the block group descriptor detail information.

badblocks

badblocks is used to check a filesystem for bad blocks. You can call it to scan for bad blocks and write a log of bad sectors by using the *-o output-file* option. When called from **e2fsck** by using the *-c* option, the bad blocks that are found will automatically be marked bad.

debugfs

With **debugfs**, you can modify the disk with direct disk writes. Since this utility is so powerful, you will normally want to invoke it as read-only until you are ready to actually make changes and write them to the disk. To invoke **debugfs** in read-only mode, do not use any switches. To open in read-write mode, add the *-w* switch. You may also want to include in the command line the device you want to work on, as in /dev/hda1 or /dev/sda1, etc. Once it is invoked, you should see a debugfs prompt.

When the superblock of a partition is damaged, you can specify a different superblock to use:

```
debugfs -b 1024 -s 8193 /dev/hda1
```

This means that the superblock at block 8193 will be used and the blocksize is 1024. Note that you have to specify the blocksize when you want to use a different superblock. The information about blocksize and backup superblocks can be found with:

```
dumpe2fs /dev/hda1
```

The first command to try after invocation, is **params** to show the mode (read-only or read-write), and the current file system. If you run this command without opening a filesystem, it will almost certainly dump core and exit. Two other commands, **open** and **close**, may be of interest when checking more than one filesystem. Close takes no argument, and appropriately enough, it closes the filesystem that is currently open. Open takes the device name as an argument. To see disk statistics from the superblock, the command **stats** will display the information by group. The command **testb** checks whether a block is in use. This can be used to test if any data is lost in the blocks marked as "bad" by the **badblocks**

command. To get the filename for a block, first use the **icheck** command to get the inode and then **ncheck** to get the filename. The best course of action with bad blocks is to mark the block "bad" and restore the file from backup.

To get a complete list of all commands, see the man page of **debugfs** or type **?**, **lr** or **list_requests**.

Prev | Up | Next

Chapter 3. Filesystem (2.203) | Home | Creating And Configuring Filesystem Options (2.203.3)

Creating And Configuring Filesystem Options (2.203.3)

The formal LPI objective states: "Candidates should be able to configure automount filesystems. This objective includes configuring automount for network and device filesystems. Also included is creating non ext2 filesystems for devices such as CD-ROMs."

Key files, terms and utilities include:

/etc/auto.master
/etc/auto.[dir]
mkisofs
dd
mke2fs

Resources: Don99; Nielsen98; Truemper00.

Autofs and automounter

Automounting is the process where mounting (and unmounting) of filesystems is done automatically by a daemon. If the filesystem is not mounted and a user tries to access it, it will be automatically (re)mounted. This is useful in networked environments (especially when not all machines are always on-line) and for removable devices, such as floppies and CD-ROMs.

The linux implementation of automounting, autofs, consists of a kernel component and a daemon called **automount**. Autofs uses **automount** to mount local and remote filesystems (over NFS) when needed and unmount them when they are not being used (after a timeout). Your /etc/init.d/autofs script first looks at /etc/auto.master:

```
# sample /etc/auto.master file
/var/autofs/floppy     /etc/auto.floppy      --timeout=2
/var/autofs/cdrom      /etc/auto.cdrom       --timeout=6
```

The file has three fields on each line. It has the directory in which all mounts will be located. Next to that is the filename of the configuration(s) for devices to be mounted. We will call these filenames the "supplemental" files. The last field displays the timeout which occurs after the given seconds of inactivity. The timeout will free or unmount all devices specified in

the supplemental files after that period of inactivity. The supplemental files can have more than one entry, but for these examples only one entry per supplemental file was used. Read below for an explanation. The supplemental files can be named anything you want them to be named.

They also have three values for each entry:

```
# sample /etc/auto.floppy file
floppy  -user,fstype=auto      :/dev/fd0
```

The first value is the "pseudo" directory. This will be explained later. The second value contains the mount options. The third value is the device (such as /dev/fd0, the floppy drive) which the "pseudo" directory is connected to.

The "pseudo" directory is contained in the directory which is defined in /etc/auto.master. When users try to access this "pseudo" directory, they will be rerouted to the device you specified. For example, if you specify a supplemental file to mount /dev/fd0 on /var/autofs/floppy/floppy, the command **ls** /var/autofs/floppy/floppy will list the contents of the floppy. But if you do the command **ls** /var/autofs/floppy, you don't see anything even though the directory /var/autofs/floppy/floppy should exist. That is because /var/autofs/floppy/floppy doesn't exist as a file or directory, but somehow the system knows that if you specifically ask for /var/autofs/floppy/floppy, it will reroute you to the floppy drive.

Now as to the reason why the floppy drive and cdrom drive are not combined into the same supplementary file. Each definition in the /etc/auto.master file will have its own **automount** program running for it. If you have several devices running on the same automount program and one of them fails, it could force the others not to work. That is why every device is running on its own automount program, which means there is one device per supplementary file per entry in the /etc/auto.master file.

CD-ROM filesystem

Creating an image for a CD-ROM

The usual utilities for creating filesystems on hard-disk partitions write an empty filesystem onto them, which is then mounted and filled with files by the users as they need it. A writable CD is only writable once so if we wrote an empty filesystem to it, it would get formatted and remain completely empty forever. This is also true for re-writable media as you cannot change arbitrary sectors yet; you must erase the whole disk. The tool to create the filesystem is called **mkisofs**. A sample usage looks like this:

```
$ mkisofs  -r   -o cd_image   private_collection/
     '---------'  '-----------------'
          |             |
```

write output to take directory as input

The option *-r* sets the permissions of all files on the CD to be public readable and enables Rockridge extensions. You probably want to use this option unless you really know what you're doing (hint: without *-r* the mount point gets the permissions of private_collection!).

mkisofs will try to map all filenames to the 8.3 format used by DOS to ensure the highest possible compatibility. In case of naming conflicts (different files have the same 8.3 name), numbers are used in the filenames and information about the chosen filename is printed via STDERR (usually the screen). Don't panic: Under Linux you will never see these odd 8.3 filenames because Linux makes use of the Rockridge extensions which contain the original file information (permissions, filename, etc.). Use the option *-J* (MS Joliet extensions) or use **mkhybrid** if you want to generate a more Windows-friendly CD-ROM. You can also use **mkhybrid** to create HFS CD-ROMS Read the man-page for details on the various options.

Reasons why the output of **mkisofs** is not directly sent to the writer device:

- mkisofs knows nothing about driving CD-writers;
- You may want to test the image before burning it;
- On slow machines it would not be reliable.

One could also think of creating an extra partition and writing the image to that partition instead to a file. This is possible, but has a few drawbacks. If you write to the wrong partition due to a typo, you could lose your complete Linux system. Furthermore, it is a waste of disk space because the CD-image is temporary data that can be deleted after writing the CD. However, using raw partitions saves you the time of deleting 650 MB of files.

Test the CD-image

Linux has the ability to mount files as if they were disk partitions. This feature is useful to check that the directory layout and file-access permissions of the CD image matches your wishes. Although media is very cheap today, the writing process is still time consuming, and you may at least want to save some time by doing a quick test.

To mount the file cd_image created above on the directory /cdrom, give the command

```
$ mount -t iso9660 -o ro,loop=/dev/loop0 cd_image /cdrom
```

Now you can inspect the files under /cdrom -- they appear exactly as they were on a real CD. To unmount the CD-image, just say **umount** /cdrom.

Write the CD-image to a CD

The command **cdrecord** is used to write images to a SCSI CD-burner. Non-SCSI writers require compatibility drivers, which make them appear as if they were real SCSI devices. For a short explanation see the section called "Configuring IDE CD burners"

Before showing you the last command, a warning that CD-writers want to be fed a constant stream of data. So, the process of writing the image to the CD must not be interrupted or a corrupt CD will result. It's easy to interrupt the data stream by deleting a very large file. Example: if you delete an old CD-image of 650 Mb, the kernel must update its information on 650,000 blocks of the hard disk (assuming you have a block size of 1 Kb for your filesystem). That takes some time and is very likely to slow down disk activity long enough for the data stream to pause for a few seconds. However, reading mail, browsing the web, or even compiling a kernel generally will not affect the writing process on modern machines.

Please note that no writer can re-position its laser and continue at the original spot on the CD when it gets disturbed. Therefore any strong vibrations or other mechanical shocks will probably destroy the CD you are writing.

Now, find the SCSI-BUS, -ID and -LUN number with **cdrecord -scanbus** and use these to write the CD:

```
shell> SCSI_BUS=0   #
shell> SCSI_ID=6    # taken from cdrecord -scanbus
shell> SCSI_LUN=0   #
shell> cdrecord -v speed=2 dev=$SCSI_BUS,$SCSI_ID,$SCSI_LUN \
         -data  cd_image

# same as above, but shorter:
shell> cdrecord -v speed=2 dev=0,6,0  -data  cd_image
```

For better readability, the coordinates of the writer are stored in three environment variables with natural names: SCSI_BUS, SCSI_ID, SCSI_LUN.

If you use cdrecord to overwrite a CD-RW, you must add the option blank=... to erase the old content. Please read the man page to learn more about the various methods of clearing the CD-RW.

If the machine is fast enough, you can feed the output of mkisofs directly into cdrecord:

```
shell> IMG_SIZE=‘mkisofs -R -q -print-size private_collection/ 2>&1 \
| sed -e "s/.* = //"‘

shell> echo $IMG_SIZE
```

```
shell> [ "0$IMG_SIZE" -ne 0 ] && mkisofs -r
private_collection/ \
| cdrecord speed=2 dev=0,6,0 tsize=${IMG_SIZE}s -data -

#                   don't forget the s --^      ^-- read data     from STDIN
```

The first command is an empty run to determine the size of the image (you need the **mkisofs** from the **cdrecord** distribution for this to work). You need to specify all parameters you will use on the final run (e.g. *-J* or *-hfs*). If your writer does not need to know the size of the image to be written, you can leave this dry run out. The printed size must be passed as a tsize-parameter to **cdrecord** (it is stored in the environment variable IMG_SIZE). The second command is a sequence of **mkisofs** and **cdrecord**, coupled via a pipe.

Making a copy of a data CD

It is possible to make a 1:1 copy of a data CD. But you should be aware of the fact that any errors while reading the original (due to dust or scratches) will result in a defective copy. Please note that both methods will fail on audio CDs!

First case: you have a CD-writer and a separate CD-ROM drive. By issuing the command

```
cdrecord -v dev=0,6,0 speed=2 -isosize /dev/scd0
```

you read the data stream from the CD-ROM drive attached as /dev/scd0 and write it directly to the CD-writer.

Second case: you don't have a separate CD-ROM drive. In this case you have to use the CD-writer to read out the CD-ROM first:

```
dd if=/dev/scd0 of=cdimage
```

This command reads the content of the CD-ROM from the device /dev/scd0 and writes it into the file cdimage. The content of this file is equivalent to what **mkisofs** produces, so you can proceed as described earlier in this document (which is to take the file cdimage as input for **cdrecord**).

Maintaining a Linux Filesystem (2.203.2)

Home

Chapter 4. Hardware (2.204)

Chapter 4. Hardware (2.204)

Prev Next

Chapter 4. Hardware (2.204)

Revision: $Revision: 1.3 $ ($Date: 2007/01/10 15:36:27 $)

This objective has a weight of 8 points and contains the following objectives:

Objective 2.204.1; Configuring RAID (2 points)

Candidates should be able to configure and implement software RAID. This objective includes using and configuring RAID 0, 1 and 5.

Objective 2.204.2; Adding New Hardware (3 points)

Candidates should be able to configure internal and external devices for a system including new hard disks, dumb terminal devices, serial UPS devices, multi-port serial cards, and LCD panels.

Objective 2.204.3; Software And Kernel Configuration (2 points)

Candidates should be able to configure kernel options to support various drives. This objective includes using LVM (Logical Volume Manager) to manage hard disk drives and partitions, as well as software tools to view and modify hard disk settings.

Objective 2.204.4; Configuring PCMCIA Devices (3 points)

Candidates should be able to configure a Linux installation to include support for mobile computer hardware extensions. This objective includes configuring those devices.

Configuring RAID (2.204.1)

Revision: $Revision: 1.10 $ ($Date: 2007/01/10 15:36:27 $)

Candidates should be able to configure and implement software RAID. This objective includes using and configuring RAID 0, 1, and 5.

Key files, terms and utilities include:

mkraid
/etc/raidtab

mdadm
mdadm.conf

Resources: LinuxRef01; Robbins01; Bar00; manpages for **mkraid** and /etc/raidtab.

What is RAID?

RAID stands for "Redundant Array of Inexpensive Disks" [1].

The basic idea behind RAID is to combine multiple small, inexpensive disk drives into an array which yields performance exceeding that of one large and expensive drive. This array of drives will appear to the computer as a single logical storage unit or drive.

RAID is a method by which information is spread across several disks, using techniques such as disk striping (RAID Level 0) and disk mirroring (RAID level 1) to achieve redundancy, lower latency and/or higher bandwidth for reading and/or writing to disk, and maximize recoverability from hard-disk crashes.

The underlying concept in RAID is that data may be distributed across each drive in the array in a consistent manner. To do this, the data much first be broken into consistently-sized chunks (often 32K or 64K in size, although different sizes can be used). Each chunk is then written to each drive in turn. When the data is to be read, the process is reversed, giving the illusion that multiple drives are actually one large drive. Primary reasons to use RAID include:

- enhanced speed
- increased storage capacity
- greater efficiency in recovering from a disk failure

RAID levels

There are a number of different ways to configure a RAID subsystem -- some maximize performance, others maximize availability, while others provide a mixture of both:

- RAID-0 (striping)
- RAID-1 (mirroring)
- RAID-4/5
- Linear mode

Level 0. RAID level 0, often called "striping", is a performance-oriented striped data mapping technique. This means the data being written to the array is broken down into strips and written across the member disks of the array. This allows high I/O performance at low inherent cost but provides no redundancy. Storage capacity of the array is equal to the total capacity of the member disks.

Level 1. RAID level 1, or "mirroring", has been used longer than any other form of RAID. Level 1 provides redundancy by writing identical data to each member disk of the array, leaving a mirrored copy on each disk. Mirroring remains popular due to its simplicity and high level of data availability. Level 1 operates with two or more disks that may use parallel access for high data-transfer rates when reading, but more commonly operate independently to provide high I/O transaction rates. Level 1 provides very good data reliability and improves performance for read-intensive applications but at a relatively high cost. Array capacity is equal to the capacity of one member disk.

Level 4. RAID level 4 uses parity concentrated on a single disk drive to protect data. It's better suited to transaction I/O rather than large file transfers. Because the dedicated parity disk represents an inherent bottleneck, level 4 is seldom used without accompanying technologies such as write-back caching. Array capacity is equal to the capacity of member disks, minus the capacity of one member disk.

Level 5. The most common type of RAID is level 5 RAID. By distributing parity across some or all of the member disk drives of an array, RAID level 5 eliminates the write bottleneck inherent in level 4. The only bottleneck is the parity calculation process. With modern CPUs and software RAID, that isn't a very big bottleneck. As with level 4, the result is asymmetrical performance, with reads substantially outperforming writes. Level 5 is often used with write-back caching to reduce the asymmetry. Array capacity is equal to the capacity of member disks, minus capacity of one member disk.

Linear RAID. Linear RAID is a simple grouping of drives to create a larger virtual drive. In linear RAID, the chunks are allocated sequentially from one member drive, going to the next drive only when the first is completely filled. This grouping provides no performance benefit, as it is unlikely that any I/O operations will be split between member drives. Linear RAID also offers no redundancy, and in fact decreases reliability -- if any one member drive fails, the entire array cannot be used. The capacity is the total of all member disks.

RAID can be implemented either in *hardware* or in *software*; both scenarios are explained below.

Hardware RAID

The hardware-based system manages the RAID subsystem independently from the host and presents to the host only a single disk per RAID array.

A typical hardware RAID device might connect to a SCSI controller and present the RAID arrays as a single SCSI drive. An external RAID system moves all RAID handling intelligence into a controller located in the external disk subsystem.

RAID controllers also come in the form of cards that act like a SCSI controller to the operating system, but handle all of the actual drive communications themselves. In these cases, you plug the drives into the RAID controller just as you would a SCSI controller, but then you add them to the RAID controller's configuration, and the operating system never

knows the difference.

Software RAID

Software RAID implements the various RAID levels in the kernel disk (block device) code. It also offers the cheapest possible solution: Expensive disk controller cards or hot-swap chassis are not required, and software RAID works with cheaper IDE disks as well as SCSI disks. With today's fast CPUs, software RAID performance can excel against hardware RAID.

Software RAID allows you to dramatically increase Linux disk IO performance and reliability without buying expensive hardware RAID controllers or enclosures. The MD driver in the Linux kernel is an example of a RAID solution that is completely hardware independent. The performance of a software-based array is dependent on the server CPU performance and load.

The concept behind Software RAID is simple -- it allows you to combine two or more block devices (usually disk partitions) into a single RAID device. So if you have three empty partitions (for example: hda3, hdb3, and hdc3), using Software RAID, you can combine these partitions and address them as a single RAID device, /dev/md0. /dev/md0 can then be formatted to contain a filesystem and used like any other partition.

Configuring RAID (using mkraid and raidstart)

Setting up RAID (in software) requires only two files to be set up under Linux: /etc/raidtab and /etc/rc.d/rc.local (or equivalent). The Linux kernel provides a special driver, /dev/md0, to access separate disk partitions as a logical RAID unit. These partitions under RAID do not actually *need* to be different disks, but in order to eliminate risks it is better to use different disks.

Follow these steps to set up RAID in Linux:

1. initialize the partitions to be used under RAID
2. configure the RAID drive
3. automate RAID activation
4. mount the filesystem on the RAID drive

Each of these steps is now described in detail:

Initialize partitions to be used in the RAID setup. Create partitions using any disk partitioning tool.

Configure the driver. To configure the driver, you need to create or edit a configuration file. By default this is the file /etc/raidtab, but **mkraid** can be told to use another file. In the configuration file you can specify the RAID-level to use, which partitions/drives to use, the device-name for the RAID drive and more. In the next section (/etc/raidtab) we will offer an

example configuration file.

Initialize RAID drive. Here's an example of the sequence of commands needed to create and format a RAID drive (mkraid):

1. initiate (/dev/md0) RAID drive: **mkraid /dev/md0**
2. create a filesystem using **mkfs**

Automate RAID activation after reboot. Run **raidstart** in one of the startup files (e.g. /etc/rc.d/rc.local or /etc/init.d/rcS) before mounting the filesystem at boot time using the following command: **raidstart** /dev/md0

mount the filesystem on the RAID drive. Edit /etc/fstab to automatically mount the filesystem on the RAID drive, or mount the filesystem manually after (manually) issuing the **raidstart** command.

mkraid

mkraid sets up a set of block devices into a single RAID array. It looks in its configuration file for the md devices mentioned on the command line, and initializes those arrays. **mkraid** works for all types of RAID arrays (RAID1, RAID4, RAID5, LINEAR and RAID0). Note that initializing RAID devices destroys all of the data on the constituent devices.

mkraid uses the file /etc/raidtab by default as its configuration file.

Persistent superblocks

By default the RAID configuration is stored in a file (i.e. /etc/raidtab). To read a file, it must be in a mounted filesystem. To mount a filesystem, the kernel must be able to recognize the disk. So initially, you could not boot from a RAID array. To solve this problem, you can activate *auto detection* in your kernel and configure **mkraid** so it will write *persistent superblocks*.

The persistent superblock is a small disk area that contains information about the RAID device. It is allocated at the end of each RAID device and helps the kernel to safely detect RAID devices - even if disks were moved between SCSI controllers. When a persistent superblock is found and the current setup does not match the administration found in there, the kernel will either correctly reorder disks, or will refuse to start up the array.

Every member of the array contains a copy of this superblock. Since this superblock carries all information necessary to start up the whole array, the array can be constructed and started if the members of that array are auto detected by the kernel. For auto detection to work this way, all the "member" RAID partitions should be marked type 0xfd (Linux RAID auto detect) e.g. by using **fdisk**. Your kernel should be configured to enable it to read persistent superblocks.

On a system that uses persistent superblocks the RAID configuration can be constructed even if the filesystem that contains /etc/raidtab was not mounted yet. Be aware: the file /etc/raidtab will not become superfluous - it is needed to be able to change and/or rebuild your configuration.

/etc/raidtab

Various Linux RAID tools use the default configuration file, /etc/raidtab.

/etc/raidtab has multiple sections, one for each md device which is being configured. Each section begins with the raiddev keyword.

> **Note**
>
> The order of items in the file is important.

Later raiddev entries can use earlier ones, and the parsing code isn't overly bright, so be sure to follow the ordering specified in the raidtab manpage for best results.

Here's a sample configuration file:

```
#
# sample raiddev configuration file
#
raiddev /dev/md0
   raid-level            0
   nr-raid-disks          2  # Specified below
   persistent-superblock   0
   chunk-size             8

   # device #1:
   #
   device               /dev/hda1
   raid-disk             0

   # device #2:
   #
   device               /dev/hdb1
   raid-disk             1

# A new section always starts with the
# keyword 'raiddev'
```

```
raiddev /dev/md1
   raid-level            5
   nr-raid-disks         3  # Specified below
   nr-spare-disks        1  # Specified below
   persistent-superblock   1
   parity-algorithm      left-symmetric

   # Devices to use in the RAID array:
   #
   device                /dev/sda1
   raid-disk             0
   device                /dev/sdb1
   raid-disk             1
   device                /dev/sdc1
   raid-disk             2

   # The spare disk:
   device                /dev/sdd1
   spare-disk            0
```

Configuring RAID using mdadm

The steps taken in using **mdadm** for RAID configuration, setup and activation are basically the same as with **mkraid** and **raidstart**. With **mdadm** however all actions are being handled by just one utility and **mdadm** doesn't necessarily need a configuration file. Read the mdadm manpage and mdadm.conf manpages for more information about mdadm, especially about create and assemble modes.

---

[1] A number of people insists that the "I" stands for "Independent"

Adding New Hardware (2.204.2)

Candidates should be able to configure internal and external devices for a system including new hard disks, dumb terminal devices, serial UPS devices, multi-port serial card, and LCD panels.

Revision: $Revision: 1.11 $ ($Date: 2004/03/24 12:50:18 $)

Key files, terms and utilities include:

XFree86
modprobe
lsmod
lsdev
lspci
setserial
usbview
/proc/bus/usb

Resources: LinuxRef02; Robbins01.2; LinuxRef03; USBRef01; LinuxRef04; LinuxRef05; Will00; the **man** pages for the various commands.

Bus structures

Most Linux installations run on PC hardware. The PC architecture currently supports a number of bus-systems that allow easy extension of hardware by plugging cards into a bus slot. Currently (2001) the PCI bus is the most widely used bus. Many newer motherboards only support PCI. On older, but fairly recent systems, often a combination of PCI and ISA slots is used, simply because there were many ISA cards available on the market. See the next section for an overview of (formerly) commonly used PC bus structures.

The main PC bus-systems

ISA

16 or 8bit, cheap, slow (usually 8MHz), standard, many cards available, but not many new motherboards are shipped with ISA anymore;

EISA

32bit, expensive, fast, few cards available, but almost obsolete.

MCA

32 or 16bit ex-IBM-proprietary, fast, obsolete/rare.

VESA-Local-Bus

32bit, based on 486 architecture, cheap, fast, many cards available, obsolete.

PCI-Local-Bus

32bit (64 bit coming), cheap, fast, many cards available, the de facto standard

We will focus on PCI, the current standard. PCI (like EISA) is not proprietary. It is faster than EISA or MCA, and cheaper. PCI is also available on other than PC hardware, e.g. DEC Alpha and PowerPC. PCI is not processor dependent. This means you can use the a PCI card in an Alpha-driven-PCI computer as well as in a Pentium-driven PCI computer, given the appropriate BIOS and software. PCI will develop into a 64-bit version and other more enhanced versions.

Plug and play

The term "Plug and Play" is often used for systems that match up devices with their device drivers and specifies their communication channels (IRQ,DMA,I/O). On the ISA bus, before Plug-and-Play, the bus-resources were set in hardware devices by jumpers. Software drivers were assigned bus-resources by configuration files (or the like) or by probing for a device at pre-determined addresses. The PCI bus was PnP-like from the beginning so it was trivial to implement PnP for this bus. Since the PCI bus specifications don't use the term "PnP" it's not clear whether or not the PCI bus should be called PnP (but it supports in hardware what today is called PnP).

Querying your PCI bus

On Linux systems you can use **lspci** to get a rundown of your PCI bus. As a normal user, you may have to type **/sbin/lspci**. In newer kernels (as of 2.1.82) **lspci** can be run as either root or any other user. **lspci** reads the /proc/bus/pci interface. The PCI ID database (e.g. /usr/share/pci.ids) is used to translate PCI vendor and device codes to readable strings.

Table 4.1. Commonly used lspci parameters

-n	shows PCI vendor and device codes as numbers instead of looking them up in the PCI ID database
-t	shows a tree-like diagram of the buses, bridges, devices and connections between them
-M	invokes busmapping mode. This will find all devices, even those that are behind misconfigured bridges, etc. This is intended for debugging only, and can crash the machine. It is only available to root.

-v verbose mode, gives detailed information about the bus and devices. Use *-vv* for even more details.

A sample output of **lspci** on a laptop follows:

```
00:00.0 Host bridge: Intel Corporation 440BX/ZX - 82443BX/ZX Host bridge (AGP disabled) (rev 03)
00:04.0 VGA compatible controller: Neomagic Corporation [MagicGraph 256AV] (rev 12)
00:05.0 Bridge: Intel Corporation 82371AB PIIX4 ISA (rev 02)
00:05.1 IDE interface: Intel Corporation 82371AB PIIX4 IDE (rev 01)
00:05.2 USB Controller: Intel Corporation 82371AB PIIX4 USB (rev 01)
00:05.3 Bridge: Intel Corporation 82371AB PIIX4 ACPI (rev 02)
00:09.0 Communication controller: Toshiba America Info Systems FIR Port (rev 23)
```

To see the interconnection between the buses, bridges and connections between them, often the command **lspci -vt** is used, e.g.

```
-[00]-+-00.0  Intel Corporation 440BX/ZX - 82443BX/ZX Host bridge (AGP disabled)
      +-04.0  Neomagic Corporation [MagicGraph 256AV]
      +-05.0  Intel Corporation 82371AB PIIX4 ISA
      +-05.1  Intel Corporation 82371AB PIIX4 IDE
      +-05.2  Intel Corporation 82371AB PIIX4 USB
      +-05.3  Intel Corporation 82371AB PIIX4 ACPI
      \-09.0  Toshiba America Info Systems FIR Port
```

In our example one PCI bus is present (00), on which 4 devices are present (00, 04, 05 and 09) and device 5 has 4 functions (05.0..05.3).

PCI latency timers

One of the things listed by the **lspci** command is the latency timer value. The latency timer can influence the performance of your PCI devices significantly. To influence the behavior of the PCI bus and devices, you can use the command **setpci**. One of the settings this command can perform is the PCI bus latency timer.

The PCI bus latency timer can range from zero to 248. If a device has a setting of zero, then it will immediately give up the bus if another device needs to transmit. If a device has a high latency setting, it will continue to use the bus for a longer period of time before stopping, while the other device waits for its turn.

If all of your devices have relatively high PCI bus latency timer settings and a lot of data is being sent over the bus, then your PCI cards are generally going to have to wait longer before they gain control of the bus, but are able to burst a lot of data across it before giving up the bus to another device. On the other hand, if all your PCI devices have low PCI-bus latency settings, then they're going to gladly give up the bus if another card needs to transmit data. This results in a much lower data-transmit latency, since no device is going to hold on to the

bus for an extended period of time, causing other devices to wait.

Low PCI-bus latency timer settings reduce the effective PCI bus bandwidth when two or more PCI devices are operating simultaneously, because large data bursts become much less frequent and control of the bus changes rapidly, increasing overhead.

Configuring PCI devices

Before you buy any PCI card, you may want to read the PCI HOWTO Will00 and the Hardware HOWTO to find out which cards are supported under Linux.

To add a PCI card to your system, you start by physically adding the PCI card. When you reboot your system, the BIOS should automatically detect the card[2].

Communication between the card and the kernel is done via the device driver for your card. Either you compile the driver into your kernel, or you compile it as a module which can be loaded using **insmod** or **modprobe**.

The documentation for your device driver will specify which parameters must be specified (if any) to the driver. Configuration of the device driver can be done either by specifying its parameters to the kernel (**lilo**, /etc/lilo.conf) or by adding them to the proper options line in /etc/conf.modules.

USB devices

As we saw in the example, our PCI bus contained a device called "USB controller". The USB (Universal Serial Bus) is a special serial device that can be used to connect peripherals of all sorts to your computer. USB can be seen as a hot-pluggable auto detecting bus: you can actively add or remove devices to and from the bus and the type of device is automatically detected. Linux supports a growing number of USB devices.

To find out which USB devices are currently connected to your system you use the command **usbview**. It is explicitly listed in the exam objectives. This command provides a graphical display of the devices/hubs connected to the system. However, **usbview** depends on X and the GTK toolkit and is not always available on all USB aware distributions. An alternate way to obtain information about your USB devices is to inspect the contents of the USB filesystem.

The USB-device filesystem is a dynamically generated filesystem. On most Linux distributions that support USB, it is automatically mounted on startup. It can be mounted almost everywhere, but the most common location is /proc/bus/usb.

To mount it "by hand", use:

```
mount -t usbdevfs none /proc/bus/usb
```

The USB device filesystem contains directories th at correspond with each USB bus. These directories are named 001, 002 etc. In addition to these, there are two generated files - the drivers and devices files.

/proc/bus/usb/drivers just lists the currently registered drivers (even if the driver is not being used by any device). This is most useful when testing module installation, and checking for USB support in an unknown kernel.

/proc/bus/usb/devices lists information about the devices currently attached to the USB bus.

```
T:  Bus=00 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
B:  Alloc= 28/900 us ( 3%), #Int=  2, #Iso=  0
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0000 ProdID=0000 Rev= 0.00

C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=255ms
T:  Bus=00 Lev=01 Prnt=01 Port=01 Cnt=01 Dev#=  2 Spd=12  MxCh= 4
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0451 ProdID=1446 Rev= 1.00

C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   1 Ivl=255ms
T:  Bus=00 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=12  MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0553 ProdID=0002 Rev= 1.00
```

The information in the /proc/bus/usb/devices output is arranged in groups. The lines starting with T: specify the USB topology. Lines that start with D: specify information from the device descriptor. Lines starting with P: contain information about the device descriptor, just like the D: lines. They are separated mainly because the information wouldn't all fit on one line. The lines that start with S:, if any, are the vendor and product strings that the device returned. The line that starts with C: give information from the configuration descriptor. The line that starts with I: is information from the interface descriptor. The line that starts with E: is information from the endpoint descriptor.

Configuring USB devices

With the release of the 2.4 kernel, Linux users gained serious USB support for a wide range of devices. The Linux USB subsystem, integrated in the kernel and supported by most Linux distributions, supports all necessary features like plug-and-play, USB bandwith allocation and more than 100 ports per bus.

Linux supports both the Universal Host Controller Interface (UHCI, used by Intel and Via motherboard chipsets) and the Open Host Controller Interface (OHCI, used by Compaq, Apple, SiS, OPTi, Lucent and ALi chipsets), making USB support available to anyone with a modern motherboard, or with a spare PCI or PcCard slot available to add in a cheap USB host controller board. Linux also supports USB hubs, which provide expansion for additional devices.

Linux 2.4 provides USB support for devices conforming to the USB Human Interface Device class, which includes USB keyboards, USB mice and touchpads, USB joysticks and USB graphics tablets. These devices appear to the computer as normal keyboards, mice and joysticks. This means that applications do not need to be updated to use the new kernel capabilities. In addition, the devices can also appear on a new "event" interface, which allows customized applications to take advantage of the additional capabilities offered by USB devices.

Configuring a new type of USB device usually requires either rebuilding the kernel or loading additional modules. Next, you'll need to create device nodes to enable communication between kernel and user programs (**mknod**). After that, the interface between kernel and userspace will be available in the form of device-files.

Additionally you'll need to configure your applications to make them aware of the newly created devices. E.g. to enable the use of USB- mice and -keyboards in an X environment, you will probably need to change sections in the XF86config file. To enable the use of USB-printers you probably will need to add an entry in /etc/printcap, etc.

Serial devices

The **setserial** command can be used to detect the configuration of your serial devices. Using the command

```
setserial /dev/ttyS0
```

will give you the the port type (i.e., 8250, 16450, 16550, 16550A, etc.), the hardware I/O port, the hardware IRQ line, its "baud base"[3] and some of its operational flags.

Multiport boards

Typical PC hardware comes with one or two serial ports. When you need to connect a larger number of serial devices, e.g. to monitor ranges of other computers or to steer industrial controllers[4], you can install a multiport adaptor, also known as multiport board or multiport card. These are plugged into your PCI or ISA bus and often have their own processing and buffering logic to ease the burden on your processor. Each multiport card has a number of external connecters (9 or 25 pin D-connectors or RJ45 "telephone" connectors). The smaller boards often have their connectors mounted directly on the board. If larger numbers of serial ports are supported, the connectors may be on the cables which come off (externally) the card (octopus cable), or can be on an external box (possibly rack mountable) which is connected by

a cable to a multiport card.

Linux will need a *device driver* to work with multiport boards. There are many types of multiport boards, some of which work in Linux, others are not. However, often someone has figured out how to make them work or wrote a driver for them. Often, the driver is implemented as a kernel-module that needs to be (automatically) loaded. As usual, certain parameters may need to be passed to the driver via **lilo**'s **append** command or via /etc/modules.conf). The manufacturer of the board should have info on their web-site and you can try kernel documentation, discussion groups and mailing list archives to obtain more information. There are kernel documentation files for Computone, Hayes-ESP, Moxa-smartio, Riscom8, Specialix, Stallion and SX (Specialix) boards.

Configuring serial devices

setserial

During the normal boot process, only COM ports 1-4 are initialized using default I/O ports and IRQ values. In order to initialize any additional serial ports, or to change the COM 1-4 ports to a non-standard configuration, the **setserial** program should be used.

setserial was created to configure the serial driver at runtime. The setserial command is most commonly executed at boot time from one of the bootscripts (e.g. 0setserial or rc.serial) This script is charged with the responsibility of initializing the serial driver to accommodate any nonstandard or unusual serial hardware in the machine.

The general syntax for the setserial command is:

```
setserial device [parameters]
```

in which the device is one of the serial devices, such as ttyS0.

Table 4.2. Commonly used setserial parameters

port [portnumber]	Specify the I/O port address in hexadecimal notation, e.g. 0x2f8
irq [irqnum]	specify which IRQ line the serial device is using
uart [type]	specify the UART type, e.g. 16550, 16450 or none (which disables the serial device)
autoirq	specify that the kernel should try to figure out the IRQ itself. Not always reliable.
skip_test	specify that the kernel should not bother with the UART type test during auto configuration (needed when the kernel has problems detecting the correct UART)

autoconfig	instructs the kernel to attempt to automatically determine the UART type located at the supplied *port*

powerd

powerd is a daemon process that sits in the background and monitors the state of an Uninterruptible Power Supply (UPS). Information comes to **powerd** from the status signals of a serial line, from a remote host or from a fifo.

Configuring disks

To install a new disk in your system, you need to physically install the hard drive and configure it in your computers BIOS. Linux will detect the new drive automatically in most cases. You could type **dmesg | more** to find out what the drive is called. The second IDE drive in your system will be named /dev/hdb. We will assume that the new drive is /dev/hdb.

You must now partition your new disk. As root in a shell type:

```
fdisk /dev/hdb
```

This will take you to a prompt that says Command (m for help):. At the prompt, type p to display the existing partitions. If you have partitions that you need to delete, type d, then at the prompt, type the number of the partition that you wish to delete. Next, type n to create the new partition. Type 1 (assuming that this will be the first partition on the new drive), and hit enter. Then you will be prompted for the cylinder that you want to start with on the drive. Next, you will be asked for the ending cylinder for the partition. Repeat these steps until all partitions are created.

To put a clean filesystem on the first partition of your newly partitioned drive, type:

```
mkfs /dev/hdb1
```

Repeat this step for all partitions. You can use the *-t* parameter to define the filesystem type to build, e.g. ext2, ext3, xfs, minix etc. (the section called "Creating Filesystems").

You will now need to decide the mount point for each new partition on your drive. We will assume /data. Type:

```
mkdir /new
```

to create the mount point for your drive. Now you will need to edit /etc/fstab. You will want to make an entry at the end of the file similar to this:

```
/dev/hdb1      /new    ext2        defaults      1  1
```

After you have created a similar entry for each partition, write the file.

```
mount -a
```

will mount the partitions in the directory that you specified in /etc/fstab.

Configuring output devices

Configuring CRT devices

On the hardware side you need to configure your graphic s card and hook-up your monitor. Next, you need to configure the X-server.

The standard X-server for Linux is XFree86. As of this writing both version 3 and 4 are widely in use. Check the XFree86 documentation to determine whether or not your chipset is supported. The suggested setup for XFree86 under Linux is a 486 or better with at least 8 megabytes of RAM, and a video card with a supported chipset. For optimal performance, you can use an accelerated card such as an S3-chipset card.

You can determine what type of chipset your video card uses by reading the card's documentation or asking the manufacturer for it. Another way to determine your chipset is by running the **SuperProbe** program included with the XFree86 distribution.

Configuring XFree86 to use your mouse, keyboard, monitor and video card correctly used to be something of a black art, requiring extensive hand-hacking of a complex configuration file.

XFree86 release 3 and later simplified this process by providing a program named **XF86Setup**. This program depends on the fact that all new PC hardware these days ships with EGA/VGA capable monitors. It invokes the VGA16 server and uses it to bring up X in a lowest-common-denominator 640x480 mode. Then it runs an interactive program that walks you through a series of five configuration panels -- mouse, keyboard, (video) card, monitor and miscellaneous server options. On Red Hat Linux, you may see a different program called **xf86config**. This works fairly similarly to **XF86Setup**, but does not itself use an X interface and the VGA16 server.

When the X-server does not come up properly or you suffer from poor quality, this is almost always due to a problem in your configuration file. Usually, the monitor timing values are off, or the videocard dotclocks are set incorrectly. Minor problems can be fixed with **xvidtune**; a really garbled screen usually means you need to go back into **XF86Setup** and choose a less-capable monitor type. If your display seems to roll or the edges are fuzzy, this is a clear indication that the monitor-timing values or dot clocks are wrong. Also be sure that you are

correctly specifying your video card chipset, as well as other options for the Device section of **XF86Config**.

You will need to hand-hack your X configuration to get optimal performance if your monitor can support 1600x1200 -- the highest canned resolution XF86Setup supports is 1280x1024. If you want to hand-hack your video configuration for this or any other reason, go see the LDP XFree86 Video Timings HOWTO LinuxRef05

Configuring LCD devices

Computers using Liquid Crystal Displays are more tricky to set up in XFree86 than ones with a normal (CRT) monitor. This is mainly due to the displays themselves: LCDs basically have a fixed resolution, although some have extra hardware built in that can cope with several different resolutions.

The standard server XF_SVGA should work in almost all cases. However, it may result in a poor picture quality, e.g. due to wrong configuration of the *Modelines* class in file /etc/X11/XF86Config . The modelines settings define the horizontal and vertical refresh rates of the electron beam in a CRT. A lot of LCD/TFT displays are compatible and are able to display a lot of common modelines correctly, however, you may need to hack the video configuration for this or some other reason LinuxRef05.

---

[2] If your system additionally contains ISA cards be sure to set up your BIOS accordingly. Consult your computers documentation to find out the proper procedure.

[3] the baud base is the clock frequency divided by 16. Normally this value is 115200, which is also the fastest baud rate which the UART can support

[4] or, as in the good old days: modems and terminals

Software And Kernel Configuration (2.204.3)

Candidates should be able to configure kernel options to support various hardware devices including UDMA66 drives and IDE CD burners. This objective includes using LVM (Logical Volume Manager) to manage hard disk drives and partitions as well as software tools to interact with hard disk settings.

Revision: $Revision: 1.11 $ ($Date: 2004/03/24 12:50:18 $)

Key files, terms and utilities include:

hdparm
tune2fs
/proc/interrupts
sysctl

Resources: Impson00; Hubert00; Colligan00; the **man** pages for the various commands.

Configuring Filesystems

The objective names the tune2fs command as one of the key utilities for this objective. The filesystem optimizing tools are handled in tune2fs.

Configuring kernel options

In the section called "Kernel Components (2.201.1)" and on, the process to configure and debug kernels is described. Additional kernel options may be configured by patching the kernel source code. Normally the kernel's tunable parameters are listed in the various header files in the source code tree. There is no golden rule to apply here - you need to read the kernel documentation or may even need to crawl through the kernel-code to find the information you need. Additionally, a running kernel can be configured by either using the /proc filesystem or by using the **sysctl** command.

Using the /proc filesystem

Current kernels also support dynamic setting of kernel parameters. The easiest method to set kernel parameters is by modifying the /proc filesystem. You can used the **echo** to set parameters, in the format:

echo "value" > /proc/kernel/parameter

e.g. to activate forwarding:

echo 1 >/proc/sys/net/ipv4/ip_forward

The changes take effect immediately but are lost during reboot.

The /proc/ filesystem can also be queried. To see which interrupts a system uses, for example you could issue

cat /proc/interrupts

which generates output that may look like this:

```
        CPU0
 0:  16313284       XT-PIC  timer
 1:    334558       XT-PIC  keyboard
 2:         0       XT-PIC  cascade
 7:     26565       XT-PIC  3c589_cs
 8:         2       XT-PIC  rtc
 9:         0       XT-PIC  OPL3-SA (MPU401)
10:         4       XT-PIC  MSS audio codec
11:         0       XT-PIC  usb-uhci
12:    714357       XT-PIC  PS/2 Mouse
13:         1       XT-PIC  fpu
14:    123824       XT-PIC  ide0
15:         7       XT-PIC  ide1
NMI:        0
```

Using sysctl

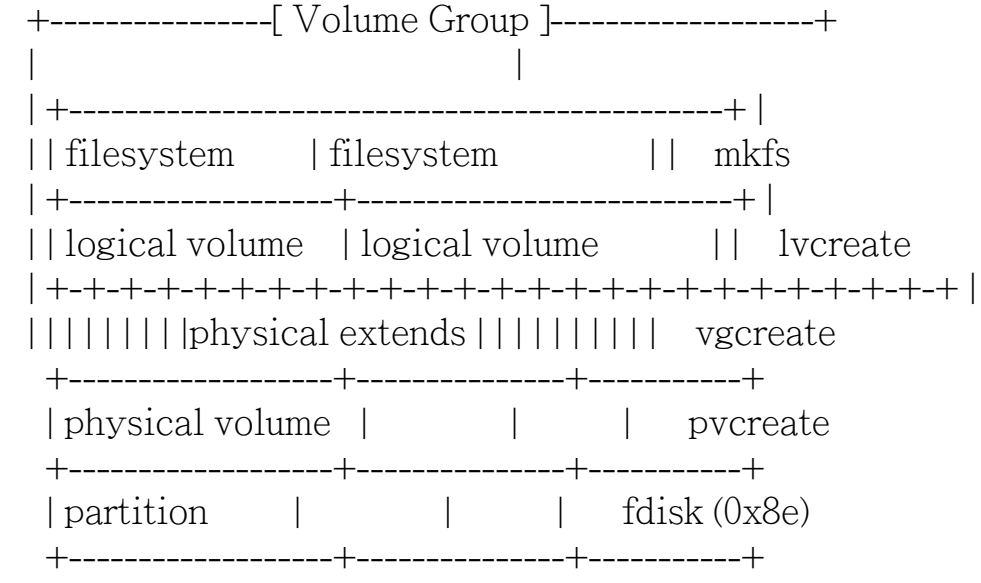
Kernel tuning can be automated by putting the **echo** commands in the startup scripts (e.g. /etc/rc.d/rc.local. Some distributions support the **sysctl** command and on those systems the initscripts run **sysctl -p** on startup. This reads the configuration in the default configuration file (/etc/sysctl.conf) and sets the tunable parameters accordingly.

Configuring Logical Volume Management

lvm is a logical volume manager for Linux. It enables you to concatenate several physical volumes (hard disks etc.) to a so-called **volume groups**, forming a storage pool, much like a virtual disk. IDE, SCSI disks, as well as, multiple devices (MD) are supported.

In the ASCII art below, the concepts/terminology used by **lvm** are sketched. On the right side the names of the commands are shown that can be used to manipulate/create the layer sketched on the left.

Figure 4.1. LVM concepts in ASCII art



The physical media / partitions

a hard disk, or a partition, e.g. /dev/hda, /dev/hda6 or /dev/sda. You should set the partition types of the disk or partition to 0x8e, which is "Linux LVM". Partitioning is done using **fdisk**. Please note that your version of **fdisk** may not yet know this type, so it will be listed as "Unknown". You can turn any consecutive number of blocks on a block device into a *Physical Volume*:

Physical Volume (PV)

a physical medium with some administrative data added to it. The command **pvcreate** can be used to add the administration onto the physical medium. The command **vgcreate** is used to create a volume group, which consists of one or more PV's. A PV that has been grouped in a volume group contains *Physical Extents*:

Physical Extents (PE)

Physical Extents are blocks of diskspace, often several megabytes in size. Using the command **lvcreate** you can assign PEs to a *Logical Volume*:

Logical Volume (LV)

A Logical Volume. On a logical volume we can use the command **mkfs** to get a *Filesystem*:

Filesystem

ext2, ReiserFS, NWFS, XFS, JFX, NTFS etc. To the linux kernel, there is no difference between a regular partition and a Logical Volume. A simple **mount** suffices to be able to use your logical volume.

Some examples of typical usage of the LVM commandset follow. Initially, you need to set the partition type for the partitions to use to create logical volumes to 0x8e. Let's assume we have partitions /dev/hda4 and /dev/hda5, and they are set to the correct partitioning type. To create a physical volume on both partitions (i.e. to set up the volume group descriptor) you type (being the superuser):

```
# pvcreate /dev/hda4 /dev/hda5
```

Now we have created two physical volumes. Next, we will create a volume group. A volume group needs to have a name (we choose volume01). To create our volume group, using our previously defined physical volumes, type:

```
# vgcreate volume01 /dev/hda5 /dev/hda4
```

The previous command line is the most basic form (refer to the manual pages for a list of configurable parameters). This will create an array of physical extents, by default they are 4 Mb in size. Using these extents we can create one or more logical volumes, e.g:

```
# lvcreate -L 100M volume01
```

.. this creates a logical volume with a default name choosen by **lvcreate** and starts with the string lvol followed by a digit -- let's assume lvol0. The logical volume will be created using the volumegroup volume01. The name of the devicefile for this volume will be /dev/volume01/lvol0. Next, we can make a filesystem on the volumegroup, as usual, using **mkfs**, e.g. an xfs filesystem:

```
# mkfs -txfs /dev/volgroup/volname
```

The resulting filesystem can be mounted as usual:

```
# mount /dev/volgroup/volname /mnt
```

One of the nicest features of LVM is the possibility of taking snapshots of volumes. A snapshot is a virtual copy of the volume to enable easy backups. Another interesting feature is striping, which can result in better performance, but also in a higher risk of losing your data.

Configuring IDE CD burners

Most newer CD burners will work with Linux. If the SCSI-version of a particular writer works, the IDE/ATAPI-version will most likely work too and vice versa. Note, however, that IDE/ATAPI writers and writers that use the parallel port require compatibility drivers, which make them appear as if they were real SCSI devices ("everything is SCSI").

In newer Linux kernels (later than 2.6), SCSI-emulation is not needed anymore.

To use IDE burners, you need to activate *both* IDE and SCSI support in your kernel, and a number of other modules, as listed below:

Description	Modulename	
Enhanced IDE/MFM/RLL...		Y
IDE/ATAPI CDROM	ide-cd	M
SCSI hostadaptor emulation	ide-scsi	M
Loopback device	loop	M
SCSI support	scsi_mod	Y/M
SCSI CD-ROM support	sr_mod	Y/M
SCSI generic support	sg	Y/M
ISO 9660 CDROM filesystem	iso9660	Y
Microsoft Joliet cdrom...	joliet	M/Y

To use your IDE burner, the modules need to be loaded. This can be done automatically via the **kerneld**/**kmod** daemons, or you need to insert the modules by hand, using **insmod** or **modprobe**. You might add the following lines to /etc/conf.modules:

```
alias   scd0 sr_mod              # load sr_mod upon access of scd0
alias   scsi_hostadaptor ide-scsi   # SCSI host adaptor emulation
options ide-cd ignore=hdb          # if /dev/hdb is your CD-writer
```

These aliases provide alternate names for the same module and are not essential, but convenient. The options provides a way to make options for module loading permanent, i.e. after you have sucessfully used them with modprobe/insmod.

Configuring harddisks using hdparm

The disk subsystem of the Linux kernel is very conservative with your data. When not completely sure if a certain disk or controller reliably handles a certain setting (such as using Ultra DMA or IDE Block Mode to transfer more sectors on a single interrupt or 32-bit bus transfers), it will default to the setting least likely to cause data corruption.

The command **hdparm** can be used to fine-tune your disk-drive. Although this utility is

intended primarily for use with (E)IDE hard disk devices, several of the options are also valid (and permitted) for use with SCSI hard disk devices and MFM/RLL hard disks with XT interfaces.

hdparm can be used with many flags that influence it's behaviour. It is mostly used to set the using_dma flag (*-d*) and the 32-bit I/O support flag (*-c*) - these two flags can help improve drive performance enormeously.

DMA (Direct Memory Access) is a technique used by newer (E)IDE drives. Regular IDE drives are interrupt driven. This means that the kernel will receive an interrupt every time a (small) amount of data needs to be transferred from disk to memory. The processor then needs to transfer the data itself. DMA (Direct Memory Access) or UDMA (Ultra DMA, a faster version) works differently: the drive is capable of addressing memory directly, hence can transfer the data into memory itself.

UltraDMA ATA66 drives are commonly used (UDMA66), as are the newer UDMA100 drives. The UDMA66 interface features burst data transfers of up to 66 MB/second, up to 8 IDE devices per system. Other interface speeds are used too: the older 33 MB/second speed (UDMA33) or UDMA100.

Recent kernels (2.4) support DMA. You need to configure the kernel to support special EIDE chipsets to be able to use all features of your controller. For older kernels, you may need to install a patch to the kernel source code first.

You can configure newer kernels (2.4) to activate DMA automatically. For older kernels, you have to activate it using the **hdparm** command. Even for newer kernels, you may want to use **hdparm** to fine-tune the disk performance, since, as stated before, the kernel will default to the setting least likely to cause data corruption. If you are sure about what you are doing, and really need the extra speed, you can fine-tune the drive using **hdparm**. For example, assuming your disk is known as /dev/hdb:

```
hdparm -X68 /dev/hdb
```

For older UDMA33 drives, use *-X66*, for UDMA100 use *-X69*. You could put a line in your rc.local file to set this automatically on (re)boot. The field of (U)DMA drives is moving constantly and fast, therefore make sure to check the kernel documentation and other references to find out about the newest developments.

Other flags (see manual page for a complete list) are:

Table 4.3. Common flags for hdparm

-a set/get the sector count for filesystem read-ahead
-A disable/enable read-lookahead

-*C* check power mode status
-*g* display the disks geometry
-*L* lock the door on removable drives (e.g. ZIP)
-*y* force standby mode
-*T* time the drive performance
-*r* set the read-only flag for the device
-*m* set/get count for multiple block IO

Here is an example of the use of **hdparm**. IF you have (E)IDE disk drives, it is often possible to boost your disk-performance by enabling 32 bit I/0 transfer mode and setting the using_dma flag for that drive. To try this, put the following line in a bootup script (e.g. /sbin/init.d/boot.local), assuming the devicename for the drive is /dev/hda:

```
hdparm -d1 -c3 /dev/hda
```

To test how much you gained type:

```
hdparm -t -T /dev/hda
```

Configuring PCMCIA Devices (2.204.4)

Candidates should be able to configure a Linux installation to include PCMCIA support. This objective includes configuring PCMCIA devices, such as ethernet adapters, to be auto detected when inserted.

Revision: $Revision: 1.10 $ ($Date: 2004/03/03 15:03:36 $)

Key files, terms and utilities include:

/etc/pcmcia/
*.opts
cardctl
cardmgr

Resources: Nielsen01; Hinds01; the **man** pages for the various commands.

Overview of PCMCIA

Nowadays, almost all laptops, and even a number of desktops, contain PCMCIA slots. PCMCIA is an abbreviation for Personal Computer Memory Card International Association, a commission that defines standards for expansion cards[5]. The PCMCIA itself was founded in 1990. It initially developed a set of standards by which memory could be added to portable systems. The PCMCIA specification 2.0 release in 1991 added protocols for I/O devices and hard disks. The 2.1 release in 1993 refined these specifications, and is the standard around which PCMCIA cards are built today. PCMCIA cards are credit card size adapters which fit into PCMCIA slots. There are three types of PCMCIA cards, Type I generally used for memory cards such as FLASH and STATIC RAM; Type II used for I/O peripherals such as serial adapters and fax-modems and Type III which are used for rotating media such as hard disks. The only difference in the physical specification for these cards is thickness: type I is the thinnest, type III the thickest.

PCMCIA cards are "hot pluggable" e.g. you can remove your network card without damaging your system (your network will not work of course) and plug it back in which will automatically start up your card and configure the card for your system. Linux supports PCMCIA standards. It features a daemon which monitors PCMCIA sockets to see if cards are removed or inserted (**cardmgr**), and runs scripts to configure the network into your system. Linux also features drivers for the various PCMCIA cards. These drivers are either part of the kernel distribution or are supplied via an additional package (Card Services for Linux).

The cardmgr

The **cardmgr** daemon is responsible for monitoring PCMCIA sockets, loading client drivers when needed and running user-level scripts in response to card insertions and removals. It records its actions in the system log, but also uses beeps to signal card status changes. The tones of the beeps indicate success or failure of particular configuration steps. Two high beeps indicate that a card was identified and configured successfully. A high beep followed by a low beep indicates that a card was identified, but could not be configured for some reason. One low beep indicates that a card could not be identified. The **cardmgr** daemon configures cards based on a database of known card types kept in /etc/pcmcia/config. This file describes the various client drivers, then describes how to identify various cards and which driver(s) belong with which cards. The format of this file is described in the **pcmcia(5)** man page.

The stab file

The **cardmgr** daemon records device information for each socket in the file /var/lib/pcmcia/stab. For the lines describing devices, the first field is the socket, the second is the device class, the third is the driver name, the fourth is used to number multiple devices associated with the same driver, the fifth is the device name, and the final two fields are the major and minor device numbers for this device (if applicable).

The /etc/pcmcia directory

The /etc/pcmcia directory contains various configuration files for PCMCIA devices. Also, it contains scripts that start or stop PCMCIA devices. For example the configuration file config.opts contains the local resource settings for PCMCIA devices, such as which ports to use, memory ranges to use and ports and irq's to exclude. Additionally, extra options for the modules can be specified here.

Card Services for Linux

"Card Services for Linux" is a complete PCMCIA or "PC Card" support package. It includes a set of loadable kernel modules that implement a version of the Card Services applications-programming interface, a set of client drivers for specific cards and a card manager daemon that can respond to card insertion and removal events, loading and unloading drivers on demand. It supports "hot swapping" of most card types, so cards can be safely inserted and ejected at any time. The release includes drivers for a variety of ethernet cards, a driver for modem and serial port cards, several SCSI adapter drivers, a driver for ATA/IDE drive cards and memory-card drivers that should support most SRAM cards and some flash cards.

You'll need the kernel source to install pcmcia-cs, since the driver modules contain references to the kernel source files. Installing the pcmcia-cs package results in a number of modules in the /lib/modules/<your-kernel-version> directory.

The PCMCIA startup script recognizes several groups of startup options which are set via environment variables. Multiple options should be separated by spaces and enclosed in quotes. Placement of startup options depends on the Linux distribution used. They may be placed directly in the startup script or they may be kept in a separate option file. These are specific for various Linux distributions.

Card Services should automatically avoid allocating IO ports and interrupts already in use by other standard devices. It will also attempt to detect conflicts with unknown devices, but this is not completely reliable. In some cases, you may need to explicitly exclude resources for a device in /etc/pcmcia/config.opts.

Newer kernels and PCMCIA

As of kernel 2.4 PCMCIA support is integrated into the kernel - that is: the modules (drivers) are part of the kernel code distribution. You may want to try that first. However, in some situations the integrated support does not work. In many cases, you will still want to download and install "Card Services for Linux" (pcmcia-cs).

The cardctl and cardinfo commands

The **cardctl** command can be used to check the status of a socket or to see how it is configured. It can also be used to alter the configuration status of a card.

Table 4.4. cardctl commands

status	Display the current socket status flags.
config	Display the socket configuration, including power settings, interrupt and I/O window settings and configuration registers.
ident	Display card identification information, including product identification strings, manufacturer ID codes and function ID codes.
suspend	Shut down and then disable power for a socket.
resume	Restore power to a socket and re-configure for use.
reset	Send a reset signal to a socket, subject to approval by any drivers already bound to the socket.
eject	Notify all client drivers that this card will be ejected, then cut power to the socket.
insert	Notify all client drivers that this card has just been inserted.
scheme	If no scheme name is given, cardctl will display the current PCMCIA configuration scheme. If a scheme name is given, cardctl will de-configure all PCMCIA devices, and reconfigure for the new scheme.

If you are running X, the **cardinfo** utility produces a graphical interface to most **cardctl**

functions.

---

[5] According to some people, PCMCIA stands for "People Cannot Memorize Computer Industry Acronyms" :-).

Prev | Up | Next

Software And Kernel Configuration (2.204.3) | Home | Chapter 5. Networking (2.205)

Chapter 5. Networking (2.205)

Prev Next

Chapter 5. Networking (2.205)

Revision: $Revision: 1.9 $ ($Date: 2004/12/01 10:46:26 $)

This objective has a weight of 8 points and contains the following objectives:

Objective 2.205.1; Basic Networking Configuration (5 points)

> The candidate should be able to configure a network device to be able to connect to a local network and a wide-area network. This objective includes being able to communicate between various subnets within a single network, configure dialup-access using mgetty, configure dialup-access using a modem or ISDN, configure authentication protocols such as PAP and CHAP and configure TCP/IP logging.

Objective 2.205.2; Advanced Network Configuration and Troubleshooting (3 points)

> The candidate should be able to configure a network device to implement various network-authentication schemes. This objective includes configuring a multi-homed network device, configuring a virtual private network and resolving networking and communication problems.

Basic Networking Configuration (2.205.1)

The candidate should be able to configure a network device to be able to connect to a local network and a wide-area network. This objective includes being able to communicate between various subnets within a single network, configure dialup access using mgetty, configure dialup access using a modem or ISDN, configure authentication protocols such as PAP and CHAP and configure TCP/IP logging.

Key files, terms and utilities include:

/sbin/route
/sbin/ifconfig
/sbin/arp
/usr/sbin/arpwatch
/etc/

Resources: Dawson00, Drake00.

Configuring the network interface

After setting up your hardware, you have to make the devices known to the kernel networking software. A couple of commands are used to configure the network interfaces and initialize the routing table. These tasks are usually performed from the network-initialization script each time you boot the system. The basic tools for this process are called **ifconfig** (where "if" stands for interface) and **route**.

ifconfig is used to make an interface accessible to the kernel networking layer. This involves the assignment of an IP address and other parameters, and activation of the interface, also known as "bringing up" the interface. Being active here means that the kernel will send and receive IP datagrams through the interface. The simplest way to invoke it is with:

ifconfig *interface ip-address*

This command assigns *ip-address* to *interface* and activates it. All other parameters are set to default values. For instance, the default network mask is derived from the network class of the IP address, such as 255.255.0.0 for a class B address.

route allows you to add or remove routes from the kernel routing table. It can be invoked as:

route {add|del} [-net|-host] *target* [*if*]

The add and del arguments determine whether to add or delete the route to *target*. The -net and -host arguments tell the route command whether the target is a network or a host (a host is assumed if you don't specify). The if argument specifies the interface and is optional, and allows you to specify to which network interface the route should be directed -- the Linux kernel makes a sensible guess if you don't supply this information.

The Loopback Interface

The very first interface to be activated is the loopback interface:

ifconfig lo 127.0.0.1

Occasionally, you will see the dummy hostname localhost being used instead of the IP address. **ifconfig** will look up the name in the hosts file, where an entry should declare it as the hostname for 127.0.0.1:

Sample /etc/hosts entry for localhost
127.0.0.1 localhost

To view the configuration of an interface, you invoke **ifconfig**, giving it only the interface name as argument:

```
$ ifconfig lo
lo        Link encap:Local Loopback inet addr:127.0.0.1
          Mask:255.0.0.0 UP LOOPBACK RUNNING MTU:3924 Metric:1 RX packets:0
          errors:0 dropped:0 overruns:0 frame:0 TX packets:0 errors:0 dropped:0
          overruns:0 carrier:0 Collisions:0
```

As you can see, the loopback interface has been assigned a netmask of 255.0.0.0, since 127.0.0.1 is a class A address.

These steps are enough to use networking applications on a stand-alone host. After adding these lines to your network initialization script[6] and making sure it will be executed at boot time, you may reboot your machine and try out various applications. For instance, **telnet localhost** should establish a telnet connection to your host, giving you a login: prompt.

The loopback interface is useful not only as an example in networking books or as a test bed during development, but is actually used by some applications during normal operation.[7] Therefore, you always have to configure it, regardless of whether your machine is attached to a network or not.

Ethernet Interfaces

Configuring an Ethernet interface is pretty much the same as the loopback interface – it just requires a few more parameters when you are using subnetting.

Suppose we have subnetted the IP network, which was originally a class B network, into class C subnetworks. To make the interface recognize this, the ifconfig incantation would look like this:

```
# ifconfig eth0 172.16.1.2 netmask 255.255.255.0
```

This command assigns the eth0 interface an IP address of 172.16.1.2. If we had omitted the netmask, ifconfig would deduce the netmask from the IP network class, which would result in an incorrect netmask of 255.255.0.0. Now a quick check shows:

```
# ifconfig eth0
eth0      Link encap 10Mps Ethernet HWaddr
          00:00:C0:90:B3:42 inet addr 172.16.1.2 Bcast 172.16.1.255 Mask
          255.255.255.0 UP BROADCAST RUNNING MTU 1500 Metric 1 RX packets 0
          errors 0 dropped 0 overrun 0 TX packets 0 errors 0 dropped 0 overrun 0
```

You can see that ifconfig automatically sets the broadcast address (the Bcast field) to the usual value, which is the host's network number with all the host bits set. Also, the maximum transmission unit (the maximum size of IP datagrams the kernel will generate for this interface) has been set to the maximum size of Ethernet packets: 1,500 bytes. The defaults are usually what you will use, but all these values can be overridden if required.

If you are using a 2.0.x or earlier kernel, you now have to install a routing entry that informs the kernel about the network that can be reached through eth0. For example:

```
# route add -net 172.16.1.0
```

At first this looks a bit like magic because it's not really clear how route detects which interface to route through. However, the trick is rather simple: the kernel checks all interfaces that have been configured so far and compares the destination address (172.16.1.0 in this case) to the network part of the interface address (that is, the bitwise *AND* of the interface address and the netmask). The only interface that matches is eth0.

Now, what's that -net option for? This is used because route can handle both routes to networks and routes to single hosts. We have to tell route explicitly that it denotes a network, so we give it the -net flag.

Routing Through a Gateway

In the previous section, we only covered the case of a host on a single Ethernet. Quite frequently, however, one encounters networks connected to one another by gateways. These gateways may simply link two or more Ethernets, but may also provide a link to the outside world, such as the Internet. In order to use a gateway, you have to provide additional routing information to the networking layer.

Imagine two ethernets linked through such a gateway, the host romeo. Assuming that romeo has already been configured, we just have to add an entry to our routing table telling the kernel all hosts on the other network can be reached through romeo. The appropriate incantation of **route** is shown below; the gw keyword tells it that the next argument denotes a gateway:

```
# route add -net 172.16.0.0 netmask 255.255.255.0 gw romeo
```

Of course, any host on the other network you wish to communicate with must have a routing entry for our network. Otherwise you would only be able to send data to the other network, but the hosts on the other network would be unable to reply.

This example only describes a gateway that switches packets between two isolated ethernets. Now assume that romeo also has a connection to the Internet (say, through an additional PPP link). In this case, we would want datagrams to any destination network to be

handed to romeo. This action can be accomplished by making it the default gateway:

```
# route add default gw romeo
```

The network name *default* is a shorthand for 0.0.0.0, which denotes the default route. The default route matches every destination and will be used if a more specific route is not available.

PPP

On Linux, PPP functionality is split into two parts: a kernel component that handles the low-level protocols (HDLC, IPCP, IPXCP, etc.) and the user space **pppd** daemon that handles the various higher-level protocols, such as PAP and CHAP. The current release of the PPP software for Linux contains the PPP daemon **pppd** and a program called **chat** that automates the dialing of the remote system.

Like SLIP, PPP is implemented by a special line discipline. To use a serial line as a PPP link, you first establish the connection over your modem as usual, and subsequently convert the line to PPP mode. In this mode, all incoming data is passed to the PPP driver, which checks the incoming HDLC frames for validity (each HDLC frame carries a 16-bit checksum), and unwraps and dispatches them. Currently, PPP is able to transport both the IP protocol, optionally using Van Jacobson header compression, and the IPX protocol.

pppd aids the kernel driver, performing the initialization and authentication phase that is necessary before actual network traffic can be sent across the link. **pppd**'s behavior may be fine-tuned using a number of options.

PPP is strictly a *peer to peer* protocol; there is (technically) no difference between the machine that dials in and the machine that is dialed into. However, for clarity's sake, it is useful to think in terms of *servers* and *clients*.

When you dial into a machine to establish a PPP connection, you are a *client*. The machine to which you connect is the *server*. When you are setting up a Linux box to receive and handle dial-in PPP connections, you are setting up a PPP *server*.

As PPP is rather complex, it is impossible to explain all of it in a single chapter. This book therefore cannot cover all aspects of **pppd**, but only gives you an introduction. For more information, consult Using & Managing PPP or the **pppd** manual pages, and README files in the pppd source distribution, which should help you sort out most questions not covered in this chapter. The PPP-HOWTO might also be of use.

Probably the greatest help you will find in configuring PPP will come from other users of the same Linux distribution. PPP configuration questions are very common, so try your local usergroup mailing list or the IRC Linux channel. If your problems persist even after reading the documentation, you could try the comp.protocols.ppp newsgroup. This is the place

where you will find most of the people involved in pppd development.

Running pppd

In this introductory example of how to establish a PPP connection with **pppd**, we will assume you are at romeo. First, dial in to the PPP server william and log in to the ppp account and william will execute its PPP driver.

PPP Client

After exiting the communications program you used for dialing, execute the following command, substituting the name of the serial device you used for the ttyS3 shown here:

```
# pppd /dev/ttyS3 38400 crtscts defaultroute
```

This command flips the serial line ttyS3 to the PPP line discipline and negotiates an IP link with william. The transfer speed used on the serial port will be 38,400 bps. The crtscts option turns on hardware handshake on the port, which is an absolute must at speeds above 9,600 bps.

The first thing **pppd** does after starting up is negotiate several link characteristics with the remote end using LCP. Usually, the default set of options **pppd** tries to negotiate will work, so we won't go into this here, except to say that part of this negotiation involves requesting or assigning the IP addresses at each end of the link.

For the time being, we also assume that william doesn't require any authentication from us, so the configuration phase is completed successfully.

pppd will then negotiate the IP parameters with its peer using IPCP, the IP control protocol. Since we didn't specify any particular IP address to **pppd** earlier, it will try to use the address obtained by having the resolver look up the local hostname. Both will then announce their addresses to each other.

Usually, there's nothing wrong with these defaults. Even if your machine is on an Ethernet, you can use the same IP address for both the Ethernet and the PPP interface. Nevertheless, **pppd** allows you to use a different address, or even to ask your peer to use some specific address. These options are discussed later.

After going through the IPCP setup phase, **pppd** will prepare your host's networking layer to use the PPP link. It first configures the PPP network interface as a point-to-point link, using ppp0 for the first PPP link that is active, ppp1 for the second, and so on. Next, it sets up a routing table entry that points to the host at the other end of the link. In the previous example, **pppd** made the default network route point to william, because we gave it the defaultroute option. The default route simplifies your routing by causing any IP datagram

destined for a nonlocal host to be sent to william; this makes sense since it is the only way it can be reached. There are a number of different routing schemes **pppd** supports, some of which we will cover later in this chapter.

Of course, the dialing can be automated using the **chat(1)** command. For more information see the manual page.

PPP Server

Running **pppd** as a server is just a matter of configuring a serial tty device to invoke **pppd** with appropriate options when an incoming data call has been received. One way to do this is to create a special account, say ppp, and give it a script or program as a login shell that invokes **pppd** with these options. Alternatively, if you intend to support PAP or CHAP authentication, you can use the **mgetty** program to support your modem and exploit its "/AutoPPP/" feature.

To build a server using the login method, you add a line similar to the following to your /etc/passwd file:

```
ppp:x:500:200:Public PPP Account:/tmp:/etc/ppp/ppplogin
```

If your system supports shadow passwords, you also need to add an entry to the /etc/shadow file:

```
ppp:!:10913:0:99999:7:::
```

Of course, the UID and GID you use depends on which user should own the connection, and how you've created it. You also have to set the password for this account using the **passwd** command.

The ppplogin script might look like this:

```
#!/bin/sh
# ppplogin - script to fire up pppd on login
mesg n
stty -echo
exec pppd -detach silent modem crtscts
```

The **mesg** command disables other users from writing to the tty by using, for example, the **write** command. The **stty** command turns off character echoing. This command is necessary; otherwise, everything the peer sends would be echoed back to it. The most important **pppd** option given is -detach because it prevents **pppd** from detaching from the controlling tty. If we didn't specify this option, it would go to the background, making the shell script exit. This in turn would cause the serial line to hang up and the connection to be

dropped. The silent option causes **pppd** to wait until it receives a packet from the calling system before it starts sending. This option prevents transmit timeouts from occurring when the calling system is slow in firing up its PPP client. The modem option makes **pppd** drive the modem control lines of the serial port. You should always turn this option on when using **pppd** with a modem. The crtscts option turns on hardware handshake.

Besides these options, you might want to force some sort of authentication, for example, by specifying auth on **pppd**'s command line or in the global options file. The manual page also discusses more specific options for turning individual authentication protocols on and off.

If you wish to use **mgetty**, all you need to do is configure **mgetty** to support the serial device your modem is connected to, by adding a line similar to the following to /etc/inittab:

```
T0:23:respawn:/sbin/mgetty ttyS0
```

Configure **pppd** for either PAP or CHAP authentication with the appropriate options in its options file, and finally, add a section similar to the following to your /etc/mgetty/login.config file:

```
# Configure mgetty to automatically detect incoming PPP calls and invoke
# the pppd daemon to handle the connection.
#
/AutoPPP/ -     ppp   /usr/sbin/pppd auth -chap +pap login
```

The first field is a special piece of magic used to detect that an incoming call is a PPP one. You must not change the case of this string; it is case sensitive. The third column ("ppp") is the user name which appears in **who** listings when someone has logged in. The rest of the line is the command to invoke. In our example, we've ensured that PAP authentication is required, disabled CHAP, and specified that the system passwd file should be used for authenticating users. This is probably similar to what you'll want. You can specify the options in the options file (see below), or on the command line if you prefer.

Here is a small checklist of tasks to perform, as well as the sequence in which they should be performed in order to have PPP dial-in working on your machine. Make sure each step works before moving on to the next:

1. Configure the modem for auto-answer mode. On Hayes-compatible modems, this is performed with a command like ATS0=3. If you're going to be using the **mgetty** daemon, this isn't necessary.
2. Configure the serial device with a **getty** type of command to answer incoming calls. A commonly used **getty** variant is **mgetty**.
3. Consider authentication. Will your callers authenticate using PAP, CHAP or system login?
4. Configure **pppd** as server as described in this section.
5. Consider routing. Will you need to provide a network route to callers? Routing can be

performed using the ip-up script.

PPP Configuration Files

Using Options Files

Before **pppd** parses its command-line arguments, it scans several files for default options. These files may contain any valid command-line arguments spread out across an arbitrary number of lines. Hash signs introduce comments.

The first options file is /etc/ppp/options, which is always scanned when **pppd** starts up. Using it to set some global defaults is a good idea, because it allows you to keep your users from doing several things that may compromise security. For instance, to make **pppd** require some kind of authentication (either PAP or CHAP) from the peer, you add the auth option to this file. This option cannot be overridden by the user, so it becomes impossible to establish a PPP connection with any system that is not in your authentication databases. Note, however, that some options can be overridden; the connect string is a good example.

The other options file, which is read after /etc/ppp/options, is .ppprc in the user's home directory. It allows each user to specify her own set of default options.

A sample /etc/ppp/options file might look like this:

```
# Global options for pppd running on romeo
lock            # use UUCP-style device locking
auth            # require authentication
usehostname     # use local hostname for CHAP
domain example.com   # our domain name
```

The lock keyword makes **pppd** comply to the standard UUCP method of device locking. With this convention, each process that accesses a serial device, say /dev/ttyS3, creates a lock file with a name like LCK..ttyS3 in a special lock-file directory to signal that the device is in use. This is necessary to prevent other programs, such as **minicom** or **uucico**, from opening the serial device while it is used by PPP.

The next three options relate to authentication and, therefore, to system security. The authentication options are best placed in the global configuration file because they are "privileged" and cannot be overridden by users' ~/.ppprc options files.

IP Configuration Options

IPCP is used to negotiate a number of IP parameters at link configuration time. Usually, each peer sends an IPCP Configuration Request packet, indicating which values it wants to

change from the defaults and the new value. Upon receipt, the remote end inspects each option in turn and either acknowledges or rejects it.

pppd gives you a lot of control over which IPCP options it will try to negotiate. You can tune it through various command-line options that we will discuss in this section.

Choosing IP Addresses

All IP interfaces require IP addresses assigned to them; a PPP device always has an IP address. The PPP suite of protocols provides a mechanism that allows the automatic assignment of IP addresses to PPP interfaces. It is possible for the PPP program at one end of a point-to-point link to assign an IP address for the remote end to use or each may use its own.

Some PPP servers that handle a lot of client sites assign addresses dynamically; addresses are assigned to systems only when calling in and are reclaimed after they have logged off. This allows the number of IP addresses required to be limited to the number of dialup lines. While limitation is convenient for managers of the PPP dialup server, it is often less convenient for users who are dialing in. In order for people to connect to your host, they must know your IP address or the hostname associated with it. If you are a user of a PPP service that assigns you an IP address dynamically, this knowledge is difficult without providing some means of allowing the DNS database to be updated after you are assigned an IP address. Here we'll cover a more preferable approach, which involves you being able to use the same IP address each time you establish your network connection.

In the previous example, we had **pppd** on romeo dial up william and establish an IP link. No provisions were made to choose a particular IP address on either end of the link. Instead, we let **pppd** take its default action. It attempts to resolve the local hostname (romeo in our example) to an IP address, which it uses for the local end, while letting the remote machine (william) provide its own. PPP supports several alternatives to this arrangement.

To ask for particular addresses, you generally provide **pppd** with the following option:

local_addr:*remote_addr*

local_addr and *remote_addr* may be specified either in dotted quad notation or as hostnames [8]. This option makes **pppd** attempt to use the first address supplied as its own IP address, and the second as the peer's. If the peer rejects either of the addresses during IPCP negotiation, no IP link will be established[9].

If you are dialing in to a server and expect it to assign you an IP address, you should ensure that **pppd** does not attempt to negotiate one for itself. To do this, use the noipdefault option and leave the *local_addr* blank. The noipdefault option will stop **pppd** from trying to use the IP address associated with the hostname as the local address.

If you want to set only the local address but accept any address the peer uses, simply leave out the *remote_addr* part. To make romeo use the IP address 130.83.4.27 instead of its own, give it 130.83.4.27: on the command line. Similarly, to set the remote address only, leave the *local_addr* field blank. By default, **pppd** will then use the address associated with your hostname.

Routing Through a PPP Link

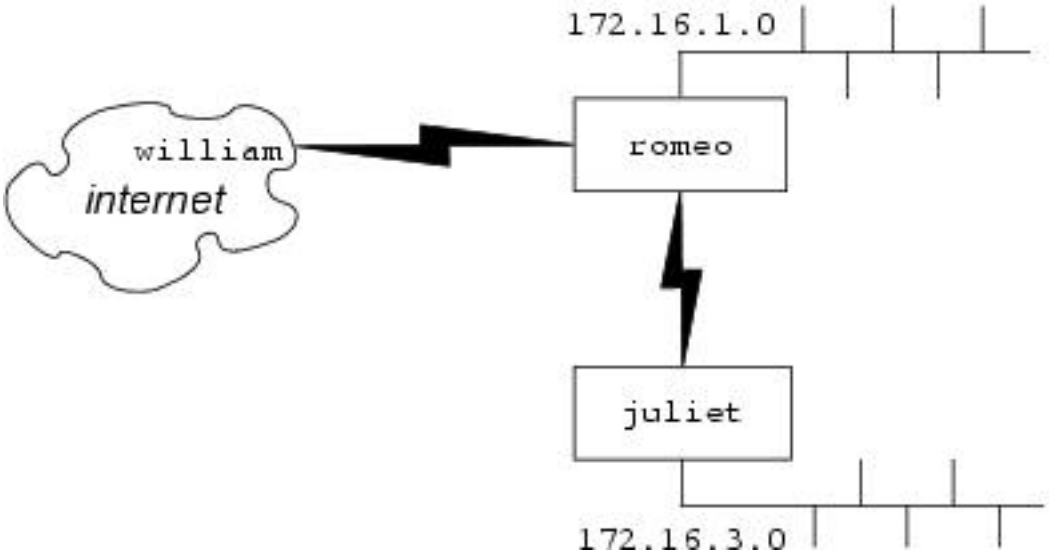
After setting up the network interface, **pppd** will usually set up a host route to its peer only. If the remote host is on a LAN, you certainly want to be able to connect to hosts *behind* your peer as well; in that case, a network route must be set up.

We have already seen that **pppd** can be asked to set the default route using the defaultroute option. This option is very useful if the PPP server you dialed-up acts as your Internet gateway.

The reverse case, in which our system acts as a gateway for a single host, is also relatively easy to accomplish. For example, let us assume a home machine is dialing in to our system (which is configured as a dialin PPP server). If we've configured our system to dynamically assign an IP address that belongs to our subnet, then we can use the proxyarp option with **pppd**, which will install a proxy ARP entry for oneshot. This automatically makes the home machine accessible from all hosts at our network.

However, things aren't always that simple. Linking two local area networks usually requires adding a specific network route because these networks may have their own default routes. Besides, having both peers use the PPP link as the default route would generate a loop through which packets to unknown destinations would ping-pong between the peers until their time-to-live expired.

Suppose you want to connect a subsidiary network with your machine romeo. The subsidiary runs an Ethernet of its own using the IP network number 172.16.3.0. The subsidiary wants to connect to your network via PPP to update customer databases. The machine romeo acts as the gateway and will support the PPP link; its peer at the new branch is called juliet and has an IP address of 172.16.3.1.



[D]

When juliet connects to romeo, it makes the default route point to romeo. On romeo, however, we will have only the point-to-point route to juliet and will have to specially configure a network route for subnet 3 that uses juliet as its gateway. We could do this manually using the **route** command after the PPP link is established, but this is not a very practical solution. Fortunately, we can configure the route automatically by using a feature of **pppd** that we haven't discussed yet; the **ip-up** command. This command is a shell script or program located in /etc/ppp that is executed by **pppd** after the PPP interface has been configured. When present, it is invoked with the following parameters:

ip-up iface device speed local_addr remote_addr

The following table summarizes the meaning of each of the arguments (in the first column, we show the number used by the shell script to refer to each argument):

Argument	Name	Purpose
$1	iface	The network interface used, e.g., ppp0
$2	device	The pathname of the serial device file used (?/dev/tty, if stdin/stdout are used)
$3	speed	The speed of the serial device in bits per second
$4	local_addr	The IP address of the remote end of the link in dotted quad notation
$5	remote_addr	The IP address of the remote end of the link in dotted quad notation

In our case, the **ip-up** script may contain the following code fragment:

```
#!/bin/sh
case $5 in
```

```
172.16.3.1)          # this is juliet
        route add -net 172.16.3.0 gw 172.16.3.1
        ;;
...
esac
exit 0
```

Similarly, /etc/ppp/ip-down can be used to undo any actions of **ip-up** after the PPP link has been taken down again. So in our /etc/ppp/ip-down script we would have a **route** command that removed the route we created in the /etc/ppp/ip-up script.

However, the routing scheme is not yet complete. We have set up routing table entries on both PPP hosts, but, so far, neither of the hosts on either network knows anything about the PPP link. This is not a big problem if all hosts at the subsidiary have their default route pointing at juliet, and all our hosts route to romeo by default. However, if this is not the case, your only option is to use a routing daemon like **gated**. After creating the network route on romeo, the routing daemon broadcasts the new route to all hosts on the attached subnets.

Authentication with PPP

With PPP, each system may require its peer to authenticate itself using one of two authentication protocols: the *Password Authentication Protocol* (PAP) or the *Challenge Handshake Authentication Protocol* (CHAP). When a connection is established, each end can request the other to authenticate itself, regardless of whether it is the caller or the called. In the description that follows, we will use the phrases "client" and "server" when we want to distinguish between the system sending authentication requests and the system responding to them. A PPP daemon on the client can ask its peer on the server for authentication by sending yet another LCP configuration request identifying the desired authentication protocol.

PAP Versus CHAP

PAP, which is offered by many Internet Service Providers, works basically the same way as the normal login procedure. The client authenticates itself by sending a username and a (optionally encrypted) password to the server, which the server compares to its secrets database[10]. This technique is vulnerable to eavesdroppers, who may try to obtain the password by listening in on the serial line, and to repeated trial and error attacks.

CHAP does not have these deficiencies. With CHAP, the server sends a randomly generated "challenge" string to the client, along with its hostname. The client uses the hostname to look up the appropriate secret, combines it with the challenge and encrypts the string using a one-way hashing function. The result is returned to the server along with the client's hostname. The server now performs the same computation and acknowledges the client if it arrives at the same result.

CHAP also doesn't require the client to authenticate itself only at startup time, but sends challenges at regular intervals to make sure the client hasn't been replaced by an intruder – for instance, by switching phone lines or because of a modem configuration error that causes the PPP daemon not to notice that the original phone call has dropped out and someone else has dialed in.

pppd keeps the secret keys for PAP and CHAP in two separate files called /etc/ppp/pap-secrets and /etc/ppp/chap-secrets. By entering a remote host in one or the other file, you have fine control over whether PAP or CHAP is used to authenticate yourself with your peer, and vice versa.

By default, **pppd** doesn't require authentication from the remote host, but it will agree to authenticate itself when requested by the remote host. Since CHAP is so much stronger than PAP, **pppd** tries to use the former whenever possible. If the peer does not support it, or if **pppd** can't find a CHAP secret for the remote system in its chap-secrets file, it reverts to PAP. If it doesn't have a PAP secret for its peer either, it refuses to authenticate altogether. As a consequence, the connection is shut down.

You can modify this behavior in several ways. When given the auth keyword, **pppd** requires the peer to authenticate itself. **pppd** agrees to use either CHAP or PAP as long as it has a secret for the peer in its CHAP or PAP database. There are other options to turn a particular authentication protocol on or off, but I won't describe them here.

If all systems you talk to with PPP agree to authenticate themselves with you, you should put the auth option in the global /etc/ppp/options file and define passwords for each system in the chap-secrets file. If a system doesn't support CHAP, add an entry for it to the pap-secrets file. That way, you can make sure no unauthenticated system connects to your host.

The next two sections discuss the two PPP secrets files, pap-secrets and chap-secrets. They are located in /etc/ppp and contain triplets of clients, servers and passwords, optionally followed by a list of IP addresses. The interpretation of the client and server fields is different for CHAP and PAP, and it also depends on whether we authenticate ourselves with the peer, or whether we require the server to authenticate itself with us.

The CHAP Secrets File

When it has to authenticate itself with a server using CHAP, **pppd** searches the chap-secrets file for an entry with the client field equal to the local hostname and the server field equal to the remote hostname sent in the CHAP challenge. When requiring the peer to authenticate itself, the roles are simply reversed: **pppd** then looks for an entry with the client field equal to the remote hostname (sent in the client's CHAP response), and the server field equal to the local hostname.

The following is a sample chap-secrets file for romeo[11]:

```
# CHAP secrets for romeo.example.com
#
# client            server              secret        addrs
#-------------------------------------------------------------------
romeo.example.com      william.shakespeare.com "Where art thou" romeo.example.com
william.shakespeare.com romeo.example.com      "To be"        william.shakespeare.com
*                      romeo.example.com      "Capulet"      pub.example.com
```

When romeo establishes a PPP connection with william, william asks romeo to authenticate itself by sending a CHAP challenge. **pppd** on romeo then scans chap-secrets for an entry with the client field equal to romeo.example.com and the server field equal to william.shakespeare.com, and finds the first line shown in the example[12]. It then produces the CHAP response from the challenge string and the secret (Use The Source Luke), and sends it off to william.

pppd also composes a CHAP challenge for william containing a unique challenge string and its fully qualified hostname (romeo.example.com). A CHAP response in the way we discussed, is formatted by william and returned to romeo. **pppd** then extracts the client hostname (william.shakespeare.com) from the response and searches the chap-secrets file for a line matching william as a client and romeo as the server. The second line does this, so **pppd** combines the CHAP challenge and the secret To be, encrypts them, and compares the result to william's CHAP response.

The optional fourth field lists the IP addresses that are acceptable for the client named in the first field. The addresses can be given in dotted quad notation or as hostnames that are looked up with the resolver. For instance, if william asks to use an IP address during IPCP negotiation that is not in this list, the request is rejected, and IPCP is shut down. In the sample file shown above, william is therefore limited to using its own IP address. If the address field is empty, any addresses are allowed; a value of - prevents the use of IP with that client altogether.

The third line of the sample chap-secrets file allows any host to establish a PPP link with romeo because a client or server field of * is a wildcard matching any hostname. The only requirements are that the connecting host must know the secret and that it must use the IP address associated with pub.example.com. Entries with wildcard hostnames may appear anywhere in the secrets file, since **pppd** will always use the best match it can find for the server/client pair.

pppd may need some help forming hostnames. As explained before, the remote hostname is always provided by the peer in the CHAP challenge or response packet. The local hostname is obtained by calling the **gethostname** function by default. If you have set the system name to your unqualified hostname, you also have to provide **pppd** with the domain name using the domain option:

```
# pppd ? domain vbrew.com
```

This provision appends our domain name to romeo for all authentication-related activities. Other options that modify **pppd**'s idea of the local hostname are usehostname and name. When you give the local IP address on the command line using *local*:*remote* and local is a name instead of a dotted quad, **pppd** uses this as the local hostname.

The PAP Secrets File

The PAP secrets file is very similar to CHAP one. The first two fields always contain a username and a hostname; the third holds the PAP secret. When the remote host sends its authentication information, **pppd** uses the entry that has a server field equal to the local hostname and a user field equal to the username sent in the request. When it is necessary for us to send our credentials to the peer, **pppd** uses the secret that has a user field equal to the local username and the server field equal to the remote hostname.

A sample PAP secrets file might look like this:

```
# /etc/ppp/pap-secrets
#
# user        server        secret        addrs
romeo-pap     william       cresspahl     romeo.example.com
william       romeo         DonaldGNUth   william.shakespeare.com
```

The first line is used to authenticate ourselves when talking to william. The second line describes how a user named william has to authenticate itself with us.

The name *romeo-pap* in the first column is the username we send to william. By default, **pppd** picks the local hostname as the username, but you can also specify a different name by giving the user option followed by that name.

When picking an entry from the pap-secrets file to identify us to a remote host, **pppd** must know the remote host's name. As it has no way of finding that out, you must specify it on the command line using the remotename keyword followed by the peer's hostname. To use the above entry for authentication with william, for example, we must add the following option to **pppd**'s command line:

```
# pppd ... remotename william user romeo-pap
```

In the fourth field of the PAP secrets file (and all following fields), you can specify what IP addresses are allowed for that particular host, just as in the CHAP secrets file. The peer will be allowed to request only addresses from that list. In the sample file, the entry that william will use when it dials in (the line where william is the client) allows it to use its real IP address and no other.

---

[6] This is usually included with your distribution.

[7] For example, all applications based on RPC use the loopback interface to register themselves with the portmapper daemon at startup. These applications include NIS and NFS.

[8] Using hostnames in this option has consequences for CHAP authentication. Please refer to the section called "The CHAP Secrets File" elsewhere in this chapter.

[9] The *ipcp-accept-local* and *ipcp-accept-remote* options instruct your **pppd** to accept the local and remote IP addresses being offered by the remote PPP, even if you've supplied some in your configuration. If these options are not configured, your **pppd** will reject any attempt to negotiate the IP addresses used.

[10] "Secret" is just the PPP name for passwords. PPP secrets don't have the same length limitation as Linux login passwords.

[11] The double quotes are not part of the secret; they merely serve to protect the whitespace within it.

[12] This hostname is taken from the CHAP challenge.

Advanced Network Configuration and Troubleshooting (2.205.2)

The candidate should be able to configure a network device to implement various network authentication schemes. This objective includes configuring a multi-homed network device, configuring a virtual private network and resolving networking and communication problems.

Key files, terms and utilities include:

/sbin/route
/bin/ifconfig
/bin/netstat
/bin/ping
/sbin/arp
/usr/sbin/tcpdump
/usr/sbin/lsof
/usr/bin/nc

Resources: Bird01, Drake00, FreeSWAN.

Virtual Private Network

What Is A VPN

VPN stands for Virtual Private Network. A VPN uses the Internet as its transport mechanism, while maintaining the security of the data on the VPN. What does a VPN do? It allows you to connect two or more remote networks securely over an insecure connection. The reason it is called *virtual* is that there is no physical connection between the two networks. Instead a non-dedicated *secure tunnel* is created through an insecure connection (like the Internet). This tunnel is generally encrypted and is only decrypted when it arrives at the destination host.

Why would you need such a network? In today's connected world, employees always need access to the data on the corporate network. Until recently this was done by dialing directly into the servers. The issue with this is that long-distance bills can be expensive as well as the cost of equipment and extra phone lines. By letting the employee connect via local internet access wherever he is, the costs are dramatically reduced.

Another reason for implementing a VPN is to integrate the LANs in several office or branches. By connecting the two networks, a user in Los Angeles can access network services in New York while still using their regular Internet connection.

VPN Types

There are many ways to implement a VPN. Many companies use proprietary software implementation while others use their routers because many routers have VPN support built in. These VPNs can be as simple as an SSH tunnel or very complicated. But all implementations create a virtual tunnel through an insecure network. Some VPN implementations include:

- IPSEC
- VPND
- SSH
- Many Cisco Routers

This book will only outline the implementations of an SSH tunnel and IPSEC.

SSH and PPP

The system described here is to implement a VPN using **ssh** and **ppp**. Basically **ssh** creates a tunnel connection which **pppd** can use to run TCP/IP traffic though. That's what makes up the VPN.

The real trick to getting **ssh** and **pppd** to play well together is the utility **pty-redir** written by Arpad Magosanyi that allows the redirection of standard in and standard out to a pseudo tty. This allows **pppd** to talk through **ssh** as if it were a serial line. On the server side, **pppd** is run as the users shell in the ssh-session, completing the link. After that, all you need to do is the routing.

The Server

The server is set up by creating an entry in /etc/passwd for a user with /usr/bin/pppd as the shell:

vpn_user:*:1004:101:VPN User,,,:/home/vpn-users:/usr/sbin/pppd

This line assumes that the user cannot login using a password. All authentication is done via **ssh**'s public key authentication system. This way, only those with keys can get in, and it's pretty much impossible to remember a binary key that's 530 characters long.

The Client

The link is created by running **pppd** through a pseudo terminal that is created by **pty-redir** and connected to **ssh**. This is done with something similar to the following sequence of commands:

```
/usr/sbin/pty-redir /usr/bin/ssh -t -e none -o 'Batchmode yes' \
        -c blowfish -i /root/.ssh/identity.vpn -l joe > /tmp/vpn-device
sleep 10

/usr/sbin/pppd `cat /tmp/vpn-device`
sleep 15

/sbin/route add -net 172.16.0.0 gw vpn-internal.mycompany.com \
                                netmask 255.240.0.0
/sbin/route add -net 192.168.0.0 gw vpn-internal.mycompany.com \
                                netmask 255.255.0.0
```

Simply, what this does is run **ssh**, redirecting its input and output to **pppd**. The options passed to **ssh** configure it to run without escape characters (-e), using the blowfish crypto algorithm (-c), using the identity file specified (-i), in terminal mode (-t), with the options *'Batchmode yes'* (-o). The **sleep** commands are used to space out the executions of the commands so that each can complete their startup before the next is run.

If the client is a standalone machine, that's all there is to it. If, on the other hand, you have more demanding routing needs (e.g., when the client is a firewall to a larger network) you will have to set up routes accordingly. On the server, this can be accomplished by running a cron job every minute that quietly sets back routes. It's not a big deal if the client isn't connected, **route** will just spit out an error (that can be sent to /dev/null.)

IPSEC

A more full-blown approach to VPN tunnels is the IPSEC protocol. IPSEC provides encryption and authentication services at the IP (Internet Protocol) level of the network protocol stack. IPSEC can be used on any machine which does IP networking. Dedicated IPSEC gateway machines can be installed wherever required to protect traffic. IPSEC can also run on routers, on firewall machines, on various application servers and on end-user desktop or laptop machines.

The basic idea of IPSEC is to provide security functions, authentication and encryption at the IP-level. This requires a higher-level protocol (IKE) to set things up for the IP-level services (ESP and AH).

Three protocols are used in an IPSEC implementation:

ESP, Encapsulating Security Payload

Encrypts and/or authenticates data;

AH, Authentication Header

Provides a packet authentication service;

IKE, Internet Key Exchange

Negotiates connection parameters, including keys, for the other two.

The term IPSEC is slightly ambiguous. In some contexts, it includes all three of the above, but, in other contexts, it refers only to AH and ESP.

FreeS/WAN is the linux implementation of IPSEC. The FreeS/WAN implementation has three main parts:

- KLIPS (kernel IPSEC) implements AH, ESP and packet handling within the kernel
- Pluto (an IKE daemon) implements IKE, negotiating connections with other systems
- various scripts provide an administrator interface to the machinery

The config file contain three parts:

one or more connection specifications

Each connection section starts with conn ident, where ident is an arbitrary name which is used to identify the connection.

connection defaults

This section starts with conn %default. For each parameter in it, any section which does not have a parameter of the same name gets a copy of the one from the *%default* section. There may be multiple %default sections, but only one default may be supplied for any specific parameter name and all *%default* sections must precede all non-*%default* sections of that type.

the config section

The config section starts with config setup and contains information used when starting the software.

A sample configuration file is shown below:

```
# basic configuration
config setup
    interfaces="ipsec0=eth0"
    klipsdebug=none
```

```
        plutodebug=none
        plutoload=%search
        plutostart=%search
        # Close down old connection when new one using same ID shows up.
        uniqueids=yes

# defaults that apply to all connection descriptions
conn %default
        # How persistent to be in (re)keying negotiations (0 means very).
        keyingtries=0
        # How to authenticate gateways
        authby=rsasig

# VPN connection for head office and branch office
conn head-branch
        # identity we use in authentication exchanges
        leftid=@head.example.com
        leftrsasigkey=0x175cffc641f...
        left=45.66.9.170
        leftnexthop=45.66.11.254
        leftsubnet=192.168.11.0/24
        # right s.g., subnet behind it, and next hop to reach left
        rightid=@branch.example.com
        rightrsasigkey=0xfc641fd6d9a24...
        right=17.120.138.134
        rightnexthop=17.120.138.1
        rightsubnet=192.168.0.0/24
        # start automatically when ipsec is loaded
        auto=start
```

To avoid trivial editing of the configuration file for each system involved in a connection, specifications are written in terms of *left* and *right* participants, rather than in terms of local and remote. Which participant is considered *left* or *right* is arbitrary; IPSEC figures out on which it is running on based on internal information. This permits identical connection specifications to be used on both ends.

The *left*, *leftsubnet* and *leftnexthop* (and the *right...* counterparts) determine the layout of the connection:

```
        subnet 192.168.11.0/24          =leftsubnet
              |
        interface 192.168.11.x
          left gateway machine
```

```
interface 45.66.9.170          =left
      |
interface 45.66.11.254         =leftnexthop
   router
interface we don't know
      |
  INTERNET
      |
interface we don't know
   router
interface 17.120.138.1         =rightnexthop
      |
interface 17.120.138.134        =right
  right gateway machine
interface 192.168.0.x
      |
subnet 192.168.0.0/24          =rightsubnet
```

The *leftid* and *leftrsasigkey* are used in authenticating the left participant. The *leftrsasigkey* is the public key of the left participant (in the example above the RSA keys are shortened for easy display). The private key is stored in the /etc/ipsec.secrets file and should be kept secure.

The IKE-daemon of IPSEC is called **pluto**. It will authenticate and negotiate the secure tunnels. Once the connections is set up, the kernel implementation of IPSEC can route traffic through the tunnel.

plutoload=%search and plutostart=%search tell the **pluto** daemon to search the configuration file for *auto=* statements. All connections with *auto=add* will be loaded in the pluto database. Connections with *auto=start* will also be started.

For more information on FreeS/WAN and IPSEC, please see the references at the beginning of this chapter.

Troubleshooting

Network troubleshooting is a very broad subject with thousands of tools available. There are some very good books available on this topic, so we will limit our discussion to an overview of some of the tools which can be used to solve or detect network problems.

ifconfig

ifconfig checks the network interface configuration. Use this command to verify the user's configuration if the user's system has been recently configured or if the user's system cannot reach the remote host while other systems on the same network can.

When **ifconfig** is entered with an interface name and no other arguments, it displays the current values assigned to that interface. For example, checking interface *eth0* system gives this report:

```
$ ifconfig eth0
eth0    Link encap:Ethernet  HWaddr 00:10:60:58:05:36
        inet addr:192.168.2.3  Bcast:192.168.2.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1398185 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1411351 errors:0 dropped:0 overruns:0 carrier:0
        collisions:829 txqueuelen:100
        RX bytes:903174205 (861.3 Mb)  TX bytes:201042106 (191.7 Mb)
        Interrupt:11 Base address:0xa000
```

The ifconfig command displays a few lines of output. The third line of the display shows the characteristics of the interface. Check for these characteristics:

UP

The interface is enabled for use. If the interface is "down", bring the interface "up" with the **ifconfig** command (e.g. **ifconfig eth0 up**).

RUNNING

This interface is operational. If the interface is not "running", the driver for this interface may not be properly installed.

The second line of **ifconfig** output shows the IP address, the subnet mask and the broadcast address. Check these three fields to make sure the network interface is properly configured.

Two common interface configuration problems are misconfigured subnet masks and incorrect IP addresses. A bad subnet mask is indicated when the host can reach other hosts on its local subnet and remote hosts on distant network, but cannot reach hosts on other local subnets. **ifconfig** quickly reveals if a bad subnet mask is set.

An incorrectly set IP address can be a subtle problem. If the network part of the address is incorrect, every **ping** will fail with the "no answer" error. In this case, using **ifconfig** will reveal the incorrect address. However, if the host part of the address is wrong, the problem can be more difficult to detect. A small system, such as a PC that only connects out to other systems and never accepts incoming connections, can run for a long time with the wrong address without its user noticing the problem. Additionally, the system that suffers the ill effects may not be the one that is misconfigured. It is possible for someone to accidentally use your IP address on his own system and for his mistake to cause your system intermittent communications problems. This type of configuration error cannot be discovered by **ifconfig**, because the error is on a remote host. The **arp** command is used for this type

of problem.

ping

The basic format of the ping command on a Linux system is:

$ ping [-c *count*] *host*

host

The hostname or IP address of the remote host being tested.

count

The number of packets to be sent in the test. Use the count field and set the value low. Otherwise, the ping command will continue to send test packets until you interrupt it, usually by pressing CTRL-C (^C).

To check that www.snow.nl can be reached from your workstation, send four packets with the following command.

```
$ ping -c 4 www.snow.nl
PING home.NL.net (193.67.79.250): 56 data bytes
64 bytes from 193.67.79.250: icmp_seq=0 ttl=245 time=32.1 ms
64 bytes from 193.67.79.250: icmp_seq=1 ttl=245 time=32.1 ms
64 bytes from 193.67.79.250: icmp_seq=2 ttl=245 time=37.6 ms
64 bytes from 193.67.79.250: icmp_seq=3 ttl=245 time=34.1 ms

--- home.NL.net ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 32.1/33.9/37.6 ms
```

This test shows a good wide-area network link to www.snow.nl with no packet loss and fast response. A small packet loss, and a round-trip time an order of magnitude higher, would not be abnormal for a connection made across a wide-area network. The statistics displayed by the ping command can indicate a low-level network problem. The key statistics are:

- The sequence in which the packets are arriving, as shown by the ICMP sequence number (icmp_seq) displayed for each packet;
- How long it takes a packet to make the round trip, displayed in milliseconds after the string time=;
- The percentage of packets lost, displayed in a summary line at the end of the ping output.

If the packet loss is high, the response time is very slow or packets are arriving out of order,

there could be a network hardware problem. If you see these conditions when communicating over great distances on a wide area network, there is nothing to worry about. TCP/IP was designed to deal with unreliable networks, and some wide-area networks suffer from a high level of packet loss. But if these problems are seen on a local-area network, they indicate trouble.

On a local-network cable segment, the round-trip time should be near 0, there should be little or no packet loss and the packets should arrive in order. If these things are not true, there is a problem with the network hardware. On an Ethernet, the problem could be improper cable termination, a bad cable segment or a bad piece of "active" hardware, such as a hub, switch or transceiver.

The results of a simple ping test, even if the ping is successful, can help direct you to further testing toward the most likely causes of the problem. But other diagnostic tools are needed to examine the problem more closely and find the underlying cause.

route

To check the routing of a linux box, the **route** command is usually entered with no parameters or with -n to turn off ip-address to name resolution. For example **route -n** might show:

```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.11.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
145.66.8.0      0.0.0.0         255.255.252.0   U     0      0        0 eth1
0.0.0.0         145.66.11.254   0.0.0.0         UG    0      0        0 eth1
```

This host has two interfaces, one on subnet 192.168.11.0/24 the other on subnet 145.66.8.0/22. There is also a default route out on *eth1* to 145.66.11.254 (denoted by the *G* under "Flags" and a "Destination" and "Genmask" of 0.0.0.0).

To be able to troubleshoot this information you need to know what the routing should be, perhaps by saving the routing information when the system is known to work.

The two most common mistakes are:

- No network entry for an interface. When a network interface is configured a routing entry should be added that informs the kernel about the network that can be reached through the interface.
- No default route (or two default routes). There should be exactly one default route. Note that two default gateways can go undetected for a long time because the routing could "accidentally" use the proper gateway.

In general, if there is a routing problem, it is better to first locate the part of the network where the problem originates, e.g. with **ping** or **traceroute** and then use **route** as part of the diagnostics.

traceroute

traceroute is a tool used to discover the links along a path. Path discovery is an essential step in diagnosing routing problems.

The **traceroute** is based on a clever use of the Time-To-Live (TTL) field in the IP packet's header. The TTL field is used to limit the life of a packet. When a router fails or is misconfigured, a routing loop or circular path may result. The TTL field prevents packets from remaining on a network indefinitely should such a routing loop occur. A packet's TTL field is decremented each time the packet crosses a router on its way through a network. When its value reaches 0, the packet is discarded rather than forwarded. When discarded, an ICMP TIME_EXCEEDED message is sent back to the packet's source to inform the source that the packet was discarded. By manipulating the TTL field of the original packet, the program **traceroute** uses information from these ICMP messages to discover paths through a network.

traceroute sends a series of UDP packets with the destination address of the device you want a path to. By default, **traceroute** sends sets of three packets to discover each hop. traceroute sets the TTL field in the first three packets to a value of 1 so that they are discarded by the first router on the path. When the ICMP TIME_EXCEEDED messages are returned by that router, **traceroute** records the source IP address of these ICMP messages. This is the IP address of the first hop on the route to the destination.

Next, three packets are sent with their TTL field set to 2. These will be discarded by the second router on the path. The ICMP messages returned by this router reveal the IP address of the second router on the path. The program proceeds in this manner until a set of packets finally has a TTL value large enough so that the packets reach their destination. Most implementations of **traceroute** default to trying only 30 hops before halting.

An example traceroute on linux looks like this:

```
$ traceroute vhost2.cistron.net
traceroute to vhost2.cistron.net (195.64.68.36), 30 hops max, 38 byte packets
 1  gateway.kabel.netvisit.nl (145.66.11.254)  56.013 ms  19.120 ms  12.642 ms
 2  145.66.3.45 (145.66.3.45)  138.138 ms  28.482 ms  28.788 ms
 3  asd-dc2-ias-ar10.nl.kpn.net (194.151.226.2)  102.338 ms  240.596 ms  253.462 ms
 4  asd-dc2-ipc-dr03.nl.kpn.net (195.190.227.242)  95.325 ms  76.738 ms  97.651 ms
 5  asd-dc2-ipc-cr01.nl.kpn.net (195.190.232.206)  61.378 ms  60.673 ms  75.867 ms
 6  asd-sara-ipc-dr01.nl.kpn.net (195.190.233.74)  111.493 ms  96.631 ms  77.398 ms
 7  asd-sara-ias-pr10.nl.kpn.net (195.190.227.6)  78.721 ms  95.029 ms  82.613 ms
 8  ams-icr-02.carrier1.net (212.4.192.60)  90.179 ms  80.634 ms  112.130 ms
```

```
 9  212.4.192.21 (212.4.192.21)  49.521 ms  80.354 ms  63.503 ms
10  212.4.194.42 (212.4.194.42)  94.528 ms  60.698 ms  103.550 ms
11  vhost2.cistron.net (195.64.68.36)  102.057 ms  62.515 ms  66.637 ms
```

Again, knowing what the route through your network should be, helps to localize the problem. Note that not all network problems can be detected with a **traceroute**, because of some complicating factors. First, the router at some hop may not return ICMP TIME_EXCEEDED messages. Second, some older routers may incorrectly forward packets even though the TTL is 0. A third possibility is that ICMP messages may be given low priority and may not be returned in a timely manner. Finally, beyond some point of the path, ICMP packets may be blocked.

The results of a **traceroute** is a great tool to narrow down the possible causes of a network problem.

arp and arpwatch

arp is used to manipulate the kernel's ARP cache. The primary options are clearing an address mapping entry and manually setting one up. For debugging purposes, the **arp** program also allows a complete dump of the ARP cache.

If you know what the ARP address of a specific host should be, the dump may be used to determine a duplicate IP-address, but running **arpwatch** on a central system might prove more effective.

arpwatch keeps track of ethernet/IP address pairings. Changes are reported via syslog and e-mail.

arpwatch will report a "changed ethernet address" when a known IP address shows up with a new ethernet address. When the old ethernet address suddenly shows up again, a "flip flop" is reported.

Detection of duplicate IP addresses is very hard even with **arpwatch**. if, for example, a rogue host uses the IP address of the host running the **arpwatch** program or never communicates with it, a duplicate IP address will go unnoticed. Still, **arpwatch** is a very useful tool in detecting networking problems.

tcpdump

tcpdump is a program that enables network administrators to inspect in real-time every packet going through the network to which his or her computer is attached. This tool is typically used to monitor active connections or troubleshoot occasional network connectivity problems.

tcpdump can generate a lot of output, so it may be useful to limit the reported packets by

specifying a port (e.g. tcpdump port 80). There are lots of other ways to specify which packets and what information we are interested in – see the manpage of **tcpdump** for more information.

nc

If the network between two hosts seems to be working, but the requested service is not, **nc** may be helpful. **nc**, or netcat, will create a TCP or UDP connection to the given host and port. Your standard in is then sent to the host and anything that comes back is sent to your standard out.

Some of the major features of netcat are:

- Outbound or inbound connections, TCP or UDP, to or from any ports
- Full DNS forward/reverse checking, with appropriate warnings
- Ability to use any local source port
- Ability to use any locally-configured network source address
- Built-in port-scanning capabilities, with randomizer
- Built-in loose source-routing capability
- Can read command line arguments from standard input
- Slow-send mode, one line every N seconds
- Hex dump of transmitted and received data
- Optional ability to let another program service establish connections
- Optional telnet-options responder

Because netcat does not make any assumptions about the protocol used across the link, it is better suited to debug connections than **telnet**.

Chapter 6. Mail & News (2.206)

Prev Next

Chapter 6. Mail & News (2.206)

Revision: $Revision: 1.8 $ ($Date: 2004/01/29 20:32:13 $)

This objective has a weight of 9 points and contains the following objectives:

Objective 2.206.1; Configuring mailing lists (1 point)

> Install and maintain mailing lists using Majordomo. Monitor Majordomo problems by viewing Majordomo logs.

Objective 2.206.2; Using Sendmail (4 points)

> Candidates should be able to manage a Sendmail configuration including email aliases, mail quotas, and virtual mail domains. This objective includes configuring internal mail relays and monitoring SMTP servers.

Objective 2.206.3 Managing Mail Traffic (3 points)

> Candidates should be able to implement client mail-management software to filter, sort and monitor incoming user mail. This objective includes using software such as procmail on both server and client side.

Objective 2.206.4; Serving News (1 point)

> Candidates should be able to install and configure news servers using inn. This objective includes customizing and monitoring served newsgroups.

Configuring mailing lists (2.206.1)

Install and maintain mailing lists using Majordomo. Monitor Majordomo problems by viewing Majordomo logs.

Key files, terms and utilities include:

Majordomo2

Installing Majordomo

Majordomo is a program which automates the management of Internet mailing lists.

Commands are sent to Majordomo via electronic mail to handle all aspects of list maintenance. Once a list is set up, virtually all operations can be performed remotely, requiring no other human intervention.

The INSTALL file contained in the distribution of Majordomo (available from the website), contains very good instructions on installing Majordomo. We will not repeat the entire INSTALL document, but only give a brief outline.

1. Create a user called "majordomo". This user should be in the group "daemon", the home directory should be the installation directory and the shell /bin/false.
2. Edit the Makefile. Define the proper locations of **perl**, the C compiler and the home directory of the "majordomo" user (i.e., the installation directory). Also set the user and group id's. Read the entire file for any other values that need to be changed.
3. Copy sample.cf to majordomo.cf and edit the latter. Most variables are self-explanatory and well-documented.
4. Type make wrapper and check for errors. This will create the wrapper program that will parse all requests made to Majordomo.
5. As root, type make install and make install-wrapper. This will install Majordomo and the wrapper.
6. Add the Majordomo-related aliases to your Sendmail aliases. This is best done by adding the following line to your sendmail.mc:

   ```
   define('ALIAS_FILE','/etc/mail/aliases,/path/to/majordomo/majordomo.aliases')
   ```

   Change /path/to/majordomo/majordomo.aliases to the real name of the Majordomo aliases file. Now add the following aliases to that file:

   ```
   majordomo: "|/path/to/majordomo/wrapper majordomo"
   owner-majordomo: you
   majordomo-owner: you
   ```

   Change "you" to your username or E-mail address.
7. Test the installation. Change to the Majordomo installation directory and, as a normal user, type ./wrapper config-test. This will test the configuration of Majordomo. Fix any errors and run again. When the process is complete, and there are no errors, config-test will offer to register your installation of Majordomo by sending information on your operating system, your Perl version and the address of the Majordomo owner to the Majordomo maintainers.

   If you configured sendmail with the restricted shell feature (smrsh), do not forget to put a link to wrapper in the smrsh directory.

Creating a Mailing list

Creating a mailing list consists of two steps: create the proper aliases for sendmail and create the necessary files for Majordomo to recognize the list. This should be done by the

maintainer of the mailinglist server (probably "root").

In the following sections, we will assume that Majordomo is installed in /usr/local/majordomo/, and that we want to create a mailinglist called "lpic2-examprep".

Aliases

Create the following aliases in either majordomo.aliases or aliases:

```
lpic2-examprep:         "|/usr/local/majordomo/wrapper resend \
                -l lpic2-examprep lpic2-examprep-list"
lpic2-examprep-list:    :include:/usr/local/majordomo/lists/lpic2-examprep
owner-lpic2-examprep:   bmesman@snow.nl
lpic2-examprep-owner:   bmesman@snow.nl
lpic2-examprep-request: "|/usr/local/majordomo/wrapper majordomo \
                -l lpic2-examprep"
lpic2-examprep-approval: bmesman@snow.nl
```

The lpic2-examprep-request alias makes it possible to send a message to this alias with a body containing "subscribe" to subscribe to the mailinglist. Without this alias, the only way to subscribe to the list is sending an E-mail to majordomo with "subscribe lpic2-examprep" in the body.

The aliases are setup in such a way that E-mail to the list will be passed through "resend" to catch Majordomo commands before passing them on to the actual list.

For more information on aliases, consult the NEWLIST file in the Majordomo distribution.

Do not forget to run the **newaliases** command, to update the actual aliases database.

Majordomo Files

In /usr/local/majordomo/lists/ create a file lpic2-examprep:

```
cd /usr/local/majordomo/lists
touch lpic2-examprep
```

This file will contain the E-mail addresses of the subscribers. Leave it empty for now.

Create the file lpic2-examprep.info with introductory info about the list:

```
echo "Discussionlist for LPIc2 exam preparation" > lpic2-examprep.info
```

The owner of the mailing list can always change the introduction with a simple E-mail (see the section called "Maintaining a Mailinglist").

Both files should be owned by user majordomo and group daemon and should be group writable.

```
chown majordomo.daemon lpic2-examprep lpic2-examprep.info
chmod 664 lpic2-examprep lpic2-examprep.info
```

Now everything is set up properly. All that remains to be done is generating a default configuration file for the mailinglist by sending an E-mail to majordomo@company.com:

```
echo "config lpic2-examprep lpic2-examprep.admin" | mail majordomo@company.com
```

Majordomo will create a default configuration file (lpic2-examprep.config) and send it to the list owner. The list owner can change the configuration (e.g., the list password!) and use the "newconfig" command to send it back.

Maintaining a Mailinglist

After a mailinglist is set up, all maintenance can be done remotely by sending an E-mail to majordomo@company.com. Maintenance commands should be put in the body of the message. The "Subject:" header will not be parsed by Majordomo. Available commands include:

lists

Show the lists available on this Majordomo server.

info <list>

Retrieve the general introductory information for the named <list>.

subscribe <list> [<address>]

Subscribe yourself (or <address> if specified) to the named <list>. Depending on the configuration of the mailinglist, you may be added directly or you may be asked to authorise the request (to prevent other people subscribing you to unwanted lists).

unsubscribe <list> [<address>]

Unsubscribe yourself (or <address> if specified) from the named <list>. "unsubscribe *" will remove you (or <address>) from all lists. This may not work if you have subscribed using multiple addresses.

intro <list>

Retrieve the introductory message, containing list policies and features. This is also

sent when you subscribe to a list.

newintro <list> <password>

Replace the information file that people get when they subscribe or do a "intro <list>". It reads everything after the **newintro** command to the end of the message or the word EOF on a line by itself as the new intro for the list.

newinfo <list> <password>

Replace the information file that people get when they do "info <list>". (This file is also sent as the "intro" if the intro file does not exist.) This reads everything after the **newinfo** command to the end of the message or the word EOF on a line by itself as the new info for the list.

passwd <list> <old_passwd> <new_passwd>

This will change the list password.

config <list> <password>

Retrieves a self-documenting configuration file for the list <list>.

newconfig <list> <password>

Validates and installs a new configuration file. It reads everything after the **newconfig** command to the end of the message or the word EOF on a line by itself as the new configuration for the list. The config file is expected to be a complete config file as returned by **config**. Incremental changing of the config file is not yet supported. As soon as the config file is validated and installed its settings are available for use. This is useful to remember if you have multiple commands in your mail message since they will be subject to the settings of the new config file. If there is an error in the config file (incorrect value...), the config file will not be accepted and an error message identifying the problem line(s) will be returned to the sender. Note that only error messages are returned to the sender, not the entire config file, so it would be a good idea to keep a copy of your outgoing email message.

Using Sendmail (2.206.2)

Candidates should be able to manage a Sendmail configuration including email aliases, mail quotas and virtual-mail domains. This objective includes configuring internal mail relays and monitoring SMTP servers.

Key files, terms and utilities include:

/etc/aliases
sendmail.cw
virtusertable
genericstable

References: the file cf.README.txt from the sendmail distribution. Also available on the web at sendmail.org.

Sendmail configuration

Sendmail uses a configuration file usually called /etc/mail/sendmail.cf. Creating this file by hand is probably the most difficult task a system administrator can encounter. Fortunately, this file can be generated by using the m4 configuration tools.

A simple mc file may look something like this:

```
divert(-1)
#
# This is a sample configuration for mailserver.company.com
#
divert(0)
include('/usr/share/sendmail/sendmail.cf/m4/cf.m4')dnl
VERSIONID(' @(#)sendmail.mc 1.0 (company.com) 19/1/02')
OSTYPE('linux')dnl
DOMAIN('company.com')dnl
MAILER(local)dnl
MAILER(smtp)dnl
```

This file consists of the following parts:

- ```
  divert(-1)
  #
  # This is a sample configuration for mailserver.company.com
  #
  divert(0)
  ```

  The divert(-1) tells m4 to ignore the following text until the next divert. This serves as a comment for your own notes.

- ```
  include('/usr/share/sendmail/sendmail.cf/m4/cf.m4')dnl
  ```

  This is the standard include file needed to generate the cf file. One of the things this will do, is set *CF_DIR*, which is the base directory for the m4 files.

- ```
  VERSIONID('@(#)sendmail.mc 1.0 (company.com) 19/1/02')
  ```

  This defines the version number of your mc file. It will create a comment header in the generated sendmail.cf for reference.

- ```
  OSTYPE('linux')dnl
  ```

  This simply includes the file $CF_DIR/ostype/linux.m4. Each operating system has its own quirks, such as the location of the alias or help files. You can see a list of the available operating system types by looking in the directory $CF_DIR/ostype/. Choose the name which best matches your operating system.

- ```
  DOMAIN('company.com')dnl
  ```

  Like the **OSTYPE()** command, the **DOMAIN(company.com)** command simply includes the contents of the file $CF_DIR/domain/company.com. This can be used to create global settings for all mc files for your domain. This line and the associated file are optional, but recommended if you are creating more than one configuration for your domain.

- ```
  MAILER(local)dnl
  MAILER(smtp)dnl
  ```

  These are the mailers to include in the sendmail configuration. In this example, we are including the local and prog mailer and the smtp family of mailers.

This is, of course, just the beginning for most configurations. You will probably want to add features by using the **FEATURE** macro. For example, the .mc line:

FEATURE('use_cw_file')

tells sendmail that you want to have it read /etc/mail/sendmail.cw as a list of alternate names for this host. The **FEATURE** may contain a single optional parameter – for example:

FEATURE('use_cw_file', 'dbm /etc/mail/alternate_names')

Some other features are:

use_cw_file

Read the file /etc/mail/sendmail.cw (also /etc/mail/local-host-names) to get alternate names for this host. This could be useful, for example, for a host MXed for a dynamic set of other hosts. If the set is static, however, just including the line "Cw<name1> <name2> ..." (where the names are fully qualified domain names) is probably better. The actual filename can be overridden by redefining confCW_FILE (e.g., "define ('confCW_FILE', '/etc/alternate_file')")

use_ct_file

Read the file /etc/mail/sendmail.ct to get the names of users who will be "trusted": that is, able to set their envelope from address using -f without generating a warning message. The actual filename can be overridden by redefining confCT_FILE.

nouucp

Don't do anything special with UUCP addresses.

always_add_domain

Include the local-host domain even on locally delivered mail. Normally it is not added on unqualified names. However, if you use a shared message store, but do not use the same user-name space everywhere, you may need the host name on local names.

masquerade_entire_domain

If masquerading is enabled (using MASQUERADE_AS) and MASQUERADE_DOMAIN is set, this feature will cause addresses to be rewritten such that the masquerading domains are hidden. All hosts within the masquerading domains will be changed to the masquerade name (used in MASQUERADE_AS). For example, if you have:

MASQUERADE_AS(masq.com)
MASQUERADE_DOMAIN(foo.org)
MASQUERADE_DOMAIN(bar.com)

then *.foo.org and *.bar.com are converted to masq.com. Without this feature, only foo.org and bar.com are masqueraded.

masquerade_envelope

This feature instructs sendmail to masquerade the envelope sender and recipient, as well as those in the headers.

genericstable

This feature will cause certain addresses originating locally (i.e., that are unqualified) or a domain listed in $=G to be looked up in a map and turned into another ("generic") form. This can change both the domain name and the user name and is similar to the userdb function. The same types of addresses as for masquerading are looked up (only header sender addresses unless the "allmasquerade" and/or "masquerade_envelope" features are invoked). Qualified addresses must have a domain in the list of names given by the macros **GENERICS_DOMAIN** or **GENERICS_DOMAIN_FILE** (analogous to **MASQUERADE_DOMAIN** and **MASQUERADE_DOMAIN_FILE**).

The argument of FEATURE('genericstable') may be the map definition; the default map definition is:

```
hash -o /etc/genericstable
```

The key for this table is either the full address or the unqualified username (the former is tried first); the value is the new user address. If the new user address does not include a domain, it will be qualified in the standard manner, i.e., using $j or the masquerade name. Note that the address to be looked up must be fully qualified. For local mail, it is necessary to use FEATURE('always_add_domain') for the addresses to be qualified.

virtusertable

This is a domain-specific form of aliasing, allowing multiple virtual domains to be hosted on one machine. If, for example, the virtuser table contained:

```
info@foo.com foo-info
info@bar.com bar-info
@baz.org jane@elsewhere.net
```

then mail addressed to info@foo.com will be sent to the address foo-info, mail addressed to info@bar.com will be delivered to bar-info, and mail addressed to anyone at baz.org will be sent to jane@elsewhere.net. The username from the original address is passed as %1 allowing:

```
@foo.org %1@elsewhere.com
```

meaning someone@foo.org will be sent to someone@elsewhere.com.

All the host names on the left-hand side (foo.com, bar.com and baz.org) must be in the list of alternate names for this host (see "use-cw-file"). The default map definition is:

hash -o /etc/virtusertable

A new definition can be specified as the second argument of the **FEATURE** macro, such as:

FEATURE('virtusertable', 'dbm -o /etc/mail/virtusers')

mailertable

A "mailer table" can be used to override routing for particular domains.

Keys in this database are fully qualified domain names or partial domains preceded by a dot – for example, "vangogh.CS.Berkeley.EDU" or ".CS.Berkeley.EDU". Values must be of the form:

mailer:domain

where "mailer" is the internal mailer name, and "domain" is where to send the message. These maps are not reflected in the message header. As a special case, the forms:

local:user

will forward to the indicated user using the local mailer,

local:

will forward to the original user in the e-mail address using the local mailer, and

error:code message

will give an error message with the indicated code and message.

smrsh

Use the SendMail Restricted SHell (smrsh) (probably provided with your distribution) instead of /bin/sh for sending mail to a program. This improves the ability of the local system administrator to control what is run via e-mail. If an argument is provided, it

is used as the pathname to smrsh; otherwise, the path defined by confEBINDIR is used for the smrsh binary – by default, /usr/lib/sm.bin/smrsh is assumed.

The commands that can be run are limited to the programs in the smrsh directory. Initial pathnames on programs are stripped, so forwarding to /usr/bin/vacation, /home/server/mydir/bin/vacation and vacation all actually forward to /usr/lib/sm.bin/vacation.

To add a program, simply create a link in /usr/lib/sm.bin/ to the program executable.

mail aliases

The mail aliases for sendmail (and many other unix mailers) are in the file /etc/mail/aliases (or /etc/aliases).

The file contains lines of the form:

name: alias1, alias2, alias3, ...

Where name is the name to alias and alias# are the aliases for that name. alias# can be another alias, a local username, a local filename, a command, an include file or an external address.

username

The username must exist on the system. If you want to be able to send E-mail to full names, you can use this to map the full names to user names:

ben.mesman: bmesman
g.g.a.m.de.vries: gerrit

/path/name

The message will be appended to the specified file. This can also be used to drop all messages to a user:

somebody: /dev/null

|command

The specified command will receive the messages on standard input. This could be used to handle mailing lists.

:include: /path/name

The names in the file /path/name will be the addresses for the alias. The filename must start with a "/". If you put (all) usernames, one per line, in a file you could have an alias:

all: :include: /etc/mail/allusers

All users in /etc/mail/allusers will receive mail for all@company.com. Note that it is not necessary to update the aliases database, with **newaliases**, when the file /etc/mail/allusers is changed.

user@domain

You can also redirect mail to an E-mail address outside your domain, by specifying the E-mail address here.

The aliases file is just the raw data file. Sendmail uses the binary file /etc/mail/aliases.db to do the alias substitution. This file is created from /etc/mail/aliases with the command **newaliases**. This command should be executed each time the aliases file is changed for the change to take effect.

After aliasing has been done, valid recipients who have a .forward file in their home directory have messages forwarded to the list of users defined in that file. When all this is done, sendmail hands the mail to the local delivery agent, usually procmail, for delivery.

Managing Mail Traffic (2.206.3)

Candidates should be able to implement client mail management software to filter, sort and monitor incoming user mail. This objective includes using software such as procmail on both server and client side.

Key files, terms and utilities include:

procmail
.procmailrc

References: McGough01 and the manual pages of **procmail** and procmailrc.

Procmail

Procmail is the default MDA (Mail Delivery Agent) for most linux distributions. When delivering mail, **procmail** first sets some environment variables to default values, reads the mail message from standard input, separates the body from the header and then starts to look for a file called $HOME/.procmailrc. According to the processing recipes in this file, the mail message gets delivered. If no .procmailrc file is found, or processing of the rc file falls off the end, **procmail** will store the mail in the default system mailbox.

The rc file can contain a mixture of environment-variable assignments (some of which have a special meaning to procmail) and recipes. In their simplest appearance, the recipes are simply one-line regular expressions that are searched for in the header of the arriving mail. The first recipe that matches is used to determine where the mail has to go (usually a file). If processing falls off the end of the rc file, procmail will deliver the mail to $DEFAULT.

By using multiple recipes, you can pre-sort your mail into several mailfolders. Bear in mind though that mail can arrive simultaneously in these mail folders (if several **procmail** programs happen to run at the same time, which is not unlikely if a lot of mail arrives). To make sure this does not result in mayhem, proper use of lock files is highly recommended.

Environment variable assignments and recipes can be freely intermixed in the rcfile. If any environment variable has a special meaning to **procmail**, it will be used appropriately the moment it is parsed (i.e., you can change the current directory whenever you want by specifying a new MAILDIR, switch lockfiles by specifying a new LOCKFILE, change the umask at any time -- the possibilities are endless).

recipes

A line starting with ":0" marks the beginning of a recipe. It has the following format:

:0 [flags] [: [locallockfile]]
<zero or more conditions (one per line)>
<exactly one action line>

Conditions start with a leading "*", everything after that character is passed on to the internal **egrep** literally, except for leading and trailing whitespace. These regular expressions are completely compatible with the normal **egrep** extended regular expressions.

Conditions are anded; if there are no conditions the result will be true by default.

A simple .procmailrc file will look something like this:

```
# Next two are needed if you invoke programs, such as
# formail, sendmail, or egrep, from your procmailrc
# SHELL=/bin/sh
# PATH=/usr/bin:/bin

# Put # before LOGFILE if you want no logging (not recommended)
LOGFILE=.procmail.log

# To insert a blank line between each message's log entry,
# uncomment next two lines (this is helpful for debugging)
# LOG="
# "

# NOTE: Upon reading the next line, Procmail does a chdir to $MAILDIR
MAILDIR=$HOME/Mail     # Make sure this directory exists!

:0:
* ^Subject:.*test
testing
```

The recipe consists of the following parts:

Notation	Meaning
:0	Begin a recipe
:	Use a lock file
*	Begin a condition

^Subject:.*test Match lines in the message.
testing Store message in the given mailbox in case matching.

To store mailing-list messages in a separate folder, use a recipe like this:

```
:0:
* ^TO_procmail@informatik\.rwth-aachen\.de
procmail
```

When you are subscribed to the procmail mailinglist, you will receive messages that are sent to the list. The ^TO_ expression will match any destination specification (To, Cc, Resent-To, etc.). The string following the ^TO_ expression is used to match (the beginning of) an E-mail address.

Procmail recipes are not case sensitive, so the above example will also match procmail@informatik.RWHT-Aachen.de. If you need to match case, you can use the "D" flag at the start of the recipe:

```
:0D:
* ^Subject:.*ADV
Potential.SPAM
```

This will only match if the "Subject:" header contains the substring "ADV" in capitals. This recipe will deliver this E-mail to the Potential.SPAM folder, because, allegedly, more respectable, unsolicited advertising has an "ADV" marker in upper case on the subject line.

Because we are not 100% sure that the substring "ADV" will always be spam, we store the message in a spam folder. We can "store" the message in /dev/null if we know that a message will always be spam:

```
:0
* ^From:.*@cyberspam\.com
/dev/null
```

This will delete all E-mail from anyone at cyberspam.com. Note that you should not use file locking in this case, hence the missing colon after ":0".

For more options, please read the man pages of **procmail** or visit the procmail web-site.

Using Sendmail (2.206.2) Home Serving news (2.206.4)

Serving news (2.206.4)

Candidates should be able to install and configure news servers using inn. This objective includes customizing and monitoring served newsgroups.

Key files, terms and utilities include:

innd

Internet News

From the README file in the INN distribution:

> INN (InterNetNews), originally written by Rich Salz, is an extremely flexible and configurable Usenet / netnews news server. For a complete description of the protocols behind Usenet and netnews, see RFC 1036 and RFC 977. In brief, netnews is a set of protocols for exchanging messages between a decentralized network of news servers. News articles are organized into newsgroups, which are themselves organized into hierarchies. Each individual news server stores locally all articles it has received for a given newsgroup, making access to stored articles extremely fast. Netnews does not require any central server; instead, each news server passes along articles it receives to all of the news servers it peers with, those servers pass the articles along to their peers, and so on, resulting in "flood fill" propagation of news articles.
>
> A news server performs three basic functions: It accepts articles from other servers and stores them on disk, sends articles it has received out to other servers and offers stored news articles to readers on demand. It additionally has to perform some periodic maintenance tasks, such as deleting older articles to make room for new ones.
>
> Originally, a news server would just store all of the news articles it had received in a filesystem. Users could then read news by reading the article files on disk (or more commonly using news reading software that did this efficiently). These days, news servers are almost always stand-alone systems and news reading is supported via network connections. A user who wants to read a newsgroup opens that newsgroup in their newsreader software, which opens a network connection to the news server and sends requests for articles and related information. The protocol that a newsreader uses to talk to a news server and that a news server uses to talk to another news server over TCP/IP is called

NNTP (Network News Transport Protocol).

INN supports accepting articles via either NNTP connections or via UUCP. **innd**, the heart of INN, handles NNTP feeding connections directly; UUCP newsfeeds use **rnews** (included in INN) to hand articles off to **innd**. Other parts of INN handle feeding articles out to other news servers, most commonly **innfeed** (for real-time outgoing feeds) or **nntpsend** and **innxmit** (used to send batches of news created by **innd** to a remote site via TCP/IP). INN can also handle outgoing UUCP feeds.

The part of INN that handles connections from newsreaders is **nnrpd**.

Also included in INN are a wide variety of supporting programs to handle periodic maintenance and recovery from crashes, process special control messages, maintain the list of active newsgroups, and generate and record a staggering variety of statistics and summary information on the usage and performance of the server.

INN also supports an extremely powerful filtering system that allows the server administrator to reject unwanted articles (such as spam and other abuses of Usenet).

Installing INN

Installing INN is not very difficult. Either download a pre-compiled binary from your favorite distribution, or download the source and compile it yourself. When compiling from source, you can probably do:

```
./configure --with-perl
make
make install
```

Read the INSTALL file to make sure this is what you want.

The difficult part is, of course, to configure INN.

Configuring INN

For the purpose of this exam preparation, we will assume that we need a stand-alone server (as opposed to distributed architectures) with a traditional spool directory. For more information on this and other configurations see Samsonova00.

Start by editing inn.conf. This file is a series of key-value pairs, one per line. First the key, followed by a colon and one or more whitespace characters and the value itself.

The standard file from the distribution is well documented, with reasonable default values. For more information on options in this file, consult the **inn.conf** manual page.

For our stand-alone configuration, make sure nnrpdposthost is set to nothing so the server knows that articles should be posted locally

Creating News Groups

Active news groups are listed in the ~news/db/active file (could be located in /var/lib/news). Editing this file by hand is not recommended, because of the rather strict syntax.

You can use **ctlinnd** to create news groups, but only when INN is running. INN will not run unless there are news groups in the active file.

To solve this chicken and egg problem, create an active file with two mandatory groups:

```
control 0000000000 0000000001 y
junk 0000000000 0000000001 y
```

Also create active.times in the same directory:

```
touch active.times
```

Make sure this file is owned by user news and group news.

If INN was installed from a DEB or RPM packet, there may already be default active and active.times files.

Now when you start INN, you can add newsgroups with the **ctlinnd** command:

```
ctlinnd newgroup <group> <flags> <creator>
```

where:

<group>

The name of the newsgroup. If the newsgroup already exists, this command will act as the changegroup command, changing the flags and/or the creator.

<flags>

Possible plags include:

y Local posting is allowed
n No local posting is allowed, only remote ones
m The group is moderated and all postings must be approved
j Articles in this group are not kept, but only passed on
x Articles cannot be posted to this newsgroup

<creator>

The name of the person creating the newsgroup. Can be omitted and will default to the newsmaster.

If you want to create a local group that can be used for testing purposes, you would issue the following command:

ctlinnd newgroup local.testing y ben

This will create a newsgroup called "local.testing" which accepts local posting.

Newsfeeds

Before we begin, it is worth mentioning the so-called wildmat pattern-matching syntax used in many configuration files. These are simple wildcard matches using the asterisk ("*") as the wildcard character, much like the simple wildcard expansion used by Unix shells.

In many cases, wildmat patterns can be specified in a comma-separated list to indicate a list of newsgroups. When used in this fashion, each pattern is checked in turn to see if it matches, and the last pattern in the line that matches the group name is used. Patterns beginning with "!" designate groups to exclude based on the pattern match. For example:

comp.*, !comp.os.*, comp.os.linux.*

In this case, we will match everything under comp ("comp.*"), except groups under comp.os ("!comp.os.*") unless they are under the comp.os.linux hierarchy ("comp.os.linux.*").

Incoming newsfeeds are configured in incoming.conf. This file should at least contain a default number of max-connections and "peer ME", which is the entry for the localhost. Without this entry articles will not appear in your spool.

An example incoming.conf file with a newsfeed from your upstream ISP for the comp.* newsgroups will look like this:

max-connections: 8 # per feed

```
peer ME {
  hostname:       "localhost, 127.0.0.1"
}

peer isp {
  hostname:       news.isp.com
  # accept the comp newsgroups from this host
  patterns:       comp.*
}
```

If you want to send the local postings to the comp.* newsgroups back to the upstream ISP, you have to set up an outgoing feed in newsfeeds:

```
news.isp.com:comp.*:Tf:news.isp.com
```

The flag "Tf" specifies that a file should be written. The name of the file must be unique, and defaults to the hostname of the feed (news.isp.com). Note that this only specifies that a log entry should be written in a file. A separate program, **innfeed**, will use this file to do the actual feed. This means that **innfeed** should be configured to read the file and send the articles to the peer. This is done by adding the following entry to the innfeed.conf file:

```
peer isp {
  hostname:       news.isp.com
}
```

Chapter 7. DNS (2.207)

Prev Next

Chapter 7. DNS (2.207)

Revision: $Revision: 1.3 $ ($Date: 2004/01/29 20:32:13 $)

This objective has a weight of 8 points and contains the following objectives:

Objective 2.207.1; Basic BIND 8 configuration (2 points)

> The candidate should be able to configure BIND to function as a caching-only DNS server. This objective includes the ability to convert a BIND 4.9 named.boot file to the BIND 8.x named.conf format and reload the DNS by using kill or ndc. This objective also includes the configuration of logging and options such as directory location for zone files.

Objective 2.207.2; Create And Maintain DNS Zones (3 points)

> The candidate should be able to create a zone file for a forward or reverse zone or root-level server. This objective includes setting appropriate values for the SOA resource record, NS records and MX records. Also included is the addition of hosts with A resource records and CNAME records, as appropriate, adding hosts to reverse zones with PTR records and adding the zone to the /etc/named.conf file using the zone statement with appropriate type, file and masters values. A candidate should also be able to delegate a zone to another DNS server.

Objective 2.207.3; Securing a DNS Server (3 points)

> The candidate should be able to configure BIND to run as a non-root user and configure BIND to run in a chroot jail. This objective includes configuring DNSSEC statements such as key and trusted-keys to prevent domain spoofing. Also included is the ability to configure a split DNS configuration using the forwarders statement and specifying a non-standard version-number string in response to queries.

Basic BIND 8 configuration (2.207.1)

Revision: $Revision: 1.5 $ ($Date: 2004/03/03 15:03:36 $)

Resources and further reading: Albitz01, DNShowto, BINDfaq.

LPIC 2 objective 207.1

The candidate should be able to configure BIND to function as a caching-only DNS server. This objective includes the ability to convert a BIND 4.9 named.boot file to the BIND 8.x named.conf format and reload the DNS by using kill or ndc. This objective also includes the configuration of logging and options such as directory location for zone files.

Key files, terms and utilities include:

/etc/named.conf
/usr/sbin/ndc
/usr/sbin/named-bootconf
kill

Name-server parts in BIND

Name servers like BIND are part of a worldwide DNS system that resolves machine names into IP-addresses.

Table Table 7.1, "Major BIND components" lists the most relevant parts of BIND software on a system. Note that directories may vary across distributions.

Table 7.1. Major BIND components

Component	Description
Program /usr/sbin/named	the real name server
Program /usr/sbin/ncd	name daemon control program
File named.conf	BIND configuration file (recent BIND)
File named.boot	obsolete BIND configuration file (BIND v4)
Program /usr/sbin/named-bootconf	converts named.boot to named.conf format
Program /etc/init.d/bind	distribution specific startfile
Directory /var/cache/bind	working directory for named

Resolving is controlled by a file nsswitch.conf which is discussed in Chapter 10, *Network Client Management (2.210)*.

BIND components will be discussed below.

The named.conf file

The file named.conf is the main configuration file of recent BIND versions (version 8 and 9 at the time of this writing). It is the first configuration file read by **named**, the DNS name daemon.

Location of named.conf

LPI lists the location of named.conf in /etc. However, the location may vary across distributions. For example in the Debian Linux distribution named.conf is located in the /etc/bind directory.

A *caching-only* named.conf file

This is an example named.conf file. The version below is taken from the Debian bind package (some comments removed).

```
options {
        directory "/var/cache/bind";

        // query-source address * port 53;

        // forwarders {
        //      0.0.0.0;
        // };
};

// reduce log verbosity on issues outside our control
logging {
        category lame-servers { null; };
        category cname { null; };
};

// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/etc/bind/db.root";
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
        type master;
        file "/etc/bind/db.local";
```

```
};

zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
        type master;
        file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
        type master;
        file "/etc/bind/db.255";
};

// add entries for other zones below here
```

The Debian bind package that contains this file, will provide a fully functional *caching-only* name server. BIND packages of other manufacturers will provide the same functionality.

A caching-only name server resolves names, which are also stored in a cache, so that they can be accessed faster when the nameserver is asked to resolve these names again. But this is what *every* name server does. The difference is this is the *only* task a *caching-only* name server performs. It does not serve out zones, except for a few internal ones.

Syntax

The named.conf file contains *statements* that start with a keyword plus a { (opening curly brace) and end with with a } (closing curly brace). The options statement is an example.

A statement may contain other statements. The forwarders statement is an example of this.

A statement may also contain things like IP addresses or file followed by a filename. These simple things *must* be terminated by a semi-colon (;).

All kinds of comments are allowed, e.g., // and # as end of line comments. See the named.conf(5) manual page for details.

Note

The ; is NOT valid as a comment sign in named.conf. It IS, however, in BIND zone files, like the file /etc/bind/db.local from the example above.

The options statement

Of the many possible entries (see named.conf(5)) inside an options statement, only the directory, forwarders, forward, version and dialup will be discussed below.

Note

There can be only *one* options statement in a named.conf file.

Some things within options

directory

Specifies the working directory for the name daemon. A common value is /var/named. Also, zone files without a directory part are looked up in this directory.

Recent distributions separate the configuration directory from the working directory. In a recent Debian Linux distribution, for example, the working directory is specified as /var/cache/bind, but all the configuration files are in /etc/bind. All zone files are also in that directory and must be specified with their directory part, as can be seen in the named.conf example above.

forwarders

The forwarders statement contains one or more IP addresses of name servers to query. How these addresses are used is specified by the forward statement described below.

The default is no forwarders. Resolving is done through the worldwide (or company local) DNS system.

Usually the specified name servers are those of the Service Provider the system connects to.

forward

The forward works only when forwarders are specified.

Two values can be specified: forward first; (default) and forward only;. With first the first query is sent to the specified name-server addresses, next queries are sent to other name servers. With only queries are limited to the specified name-server addresses.

An example with both forwarders and forward:

```
options {
  // ...

  forwarders {
    123.12.134.2;
    123.12.134.3;
  }

  forward only;

  // ...
}
```

In this example bind is told to query *only* the name servers 123.12.134.2 and 123.12.134.3.

version

It is possible to query the version from a running name server:

```
dig txt chaos version.bind
```

Note that the BIND version is shown in the output. Because some BIND versions have known exploits, the BIND version is sometimes kept hidden. The version specification:

```
version "not revealed";
```

inside the options statement leads to not revealed responses on version queries.

dialup

When a name server sits behind a firewall, that connects to the outside world through a dialup connection, some maintenance normally done by name servers might be unwanted.

The following example, also inside the options part, stops external zone maintenance:

```
heartbeat-interval 0;
dialup yes;
```

Many more options can be placed inside the options block. See the manual page.

The logging statement

The BIND (version 8 and 9) logging system is too elaborate to discuss in detail here. The

distinction between *categories* and *channels* is important.

A *channel* is an output specification. The null channel, for example, dismisses any output sent to the channel.

A *category* is a type of data. The category security is one of many categories. To log messages of type (*category*) security, for example, to the default_syslog channel, use the following:

```
logging {
   category security { default_syslog; };
   // ...
};
```

To turn off logging for certain types of data, send it to the null channel, as is done in the example named.conf shown earlier:

```
logging {
      category lame-servers { null; };
      category cname { null; };
};
```

This means that messages of types lame-servers and cname are being discarded.

There are *reasonable defaults* for logging. This means that a named.conf without logging statement is possible.

Note

A maximum of *one* logging statement is allowed in a named.conf file.

Predefined zone statements

A zone defined in named.conf will be available as @ inside the corresponding zone file. The @ is called the *current origin*. For example,

```
zone "127.in-addr.arpa" {
      type master;
      file "/etc/bind/db.127";
};
```

will result in a *current origin* of 127.in-addr.arpa that is available as @ in file etc/bind/db.127.

Details about zone files, as well as how to create your own zone files and statements will be covered in the section called "Create And Maintain DNS Zones (2.207.2)".

Converting BIND v4 to BIND v8 configuration

To convert the BIND version 4 configuration file named.boot to named.conf (the configuration file for BIND versions 8 and 9) run the script named-bootconf that comes with the BIND distribution.

> **Note**
>
> As of BIND version 8.2, zone files must start with an extra $TTL field. This is discussed in the section called "The $TTL statement".

The named name server daemon

The named name server daemon is the program that communicates with other name servers to resolve names. It accepts queries, looks in its cache and queries other name servers if it does not yet have the answer.

Once it finds an answer in its own cache or database or receives an answer from another nameserver, it sends the answer back to the name server that sent the query in the first place.

Table Table 7.2, " Controlling named " lists ways to control the named name server.

Table 7.2. Controlling named

Method	See
The ndc program	the section called "The ndc program"
Sending signals	the section called " Sending signals to named "
Using a start/stop script	the section called " Controlling named with a start/stop script "

The ndc program

The ndc (name daemon control) program can be used to control the named name server daemon. It can be used in batch- or interactive mode.

In *batchmode* a *command* is specified as a parameter to ndc:

```
ndc reload
```

This will ask the name server to reload its configuration and zone files. All *commands* specified this way are understood by the name daemon. The special command help will show a list of possible commands.

The *interactive* mode is entered when **ndc** does not see a command on the command line. It presents a prompt. Now, there are two types of commands: commands understood by the name server and commands to control **ndc**.

commands understood by the name server. The help command presents a list of commands understood by the name server. These are the same commands that can be specified as parameters on the command line.

commands to control ndc. commands to control ndc itself. These start with a slash (/). The /h command shows a list; the /e exits ndc.

There is much more to know about **ndc**. See the manpage.

Sending signals to named

It is possible to send signals to the named process to control its behavior. A full list of signals can be found in the named manpage. One example is the SIGHUP signal, that causes named to reload named.conf and the database files.

Signals are sent to named with the kill command, e.g.,

```
kill -HUP 217
```

This sends a SIGHUP signal to a named process with process id 217, which triggers a reload.

Controlling named with a start/stop script

Most distributions will come with a start/stop script that allows you to start, stop or control named manually, e.g., /etc/init.d/bind in Debian.

Table Table 7.3, "/etc/init.d/bind parameters" lists parameters that a current version of /etc/init.d/bind accepts.

Table 7.3. /etc/init.d/bind **parameters**

Parameter	Description

start	starts named
stop	stops named
restart	stops and restarts named
reload	reloads configuration
force-reload	same as restart

Create And Maintain DNS Zones (2.207.2)

Revision: $Revision: 1.5 $ ($Date: 2004/01/29 20:32:13 $)

Resources and further reading: Albitz01, DNShowto, BINDfaq.

LPIC 2 objective 207.2

The candidate should be able to create a zone file for a forward or reverse zone or root-level server. This objective includes setting appropriate values for the SOA resource record, NS records and MX records. Also included is the addition of hosts with A resource records and CNAME records, as appropriate, adding hosts to reverse zones with PTR records and adding the zone to the /etc/named.conf file using the zone statement with appropriate type, file and masters values. A candidate should also be able to delegate a zone to another DNS server.

Key files, terms and utilities include:

contents of /var/named
zone file syntax
resource record formats
dig
nslookup
host

Zones and reverse zones

There are *zones* and *reverse zones*. Each named.conf will contain definitions for both.

Examples of zones are localhost (used internally) and example.com (an external example which does not necessarily exist in reality). Examples of reverse zones are 127.in-addr.arpa (used internally), and 240.123.224.in-addr.arpa (a real-world example).

How zones are related to reverse zones is shown below.

The db.local file

A special domain, localhost, will be predefined in most cases.

Here is the corresponding zone file /etc/bind/db.local called from the example named.conf shown earlier in this chapter.

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     localhost. root.localhost. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      localhost.
@       IN      A       127.0.0.1
```

The @ contains the name of the zone. It is called the *current origin*. A zone statement in named.conf defines that *current origin*, as is seen in this part of the named.conf file we saw earlier:

```
zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};
```

So in this case the zone is called localhost and all current origins in the zone file will become localhost.

Other parts of a zone file will be explained in the section called "Zone files" below.

The db.127 file

To each IP range in a zone file, there is a corresponding *reverse zone* that is described in a *reverse zone file*. Here is the file /etc/bind/db.127:

```
;
; BIND reverse data file for local loopback interface
;
$TTL    604800
@       IN      SOA     localhost. root.localhost. (
                              1         ; Serial
```

```
        604800     ; Refresh
         86400     ; Retry
       2419200     ; Expire
        604800 )   ; Negative Cache TTL
;
@       IN  NS     localhost.
1.0.0   IN  PTR    localhost.
```

This is the calling part from named.conf:

```
zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};
```

As can be seen, the *current origin* will be 127.in-addr.arpa, so all @'s in the reverse zone file will be replaced by 127.in-addr.arpa.

But there is more: all host names that do not end in a dot get the current origin appended. This is **important** , so I repeat:

> all *host names that do not end in a dot* get the *current origin* appended.

As an example: 1.0.0 will become 1.0.0.127.in-addr.arpa.

Normal IP addresses (e.g. 127.0.0.1) do not get the current origin appended.

Again, details about the reverse zone file will be discussed below.

The hints file

The localhost and 127.in-addr.arpa zones are for internal use within a system.

In the outside world, zones are hierarchically organized. The root zone (denoted with a dot (.)) is listed in a special hints file. The following zone statement from named.conf reads a root zone file called /etc/bind/db.root

```
zone "." {
    type hint;
    file "/etc/bind/db.root";
};
```

By the way, note the type: hint! It is a special type for the root zone. Nothing else is stored

in this file, and it is not updated dynamically.

Here is a part from the db.root file.

```
; formerly NS.INTERNIC.NET
;
.                       3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000      A     198.41.0.4
;
; formerly NS1.ISI.EDU
;
.                       3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.     3600000      A     128.9.0.107
```

Note the dot at the left, this is the root zone!

Either your distribution or you personally must keep the root zone file current. You can look at ftp.rs.internic.net for a new version. Or, you could run something like

```
dig @a.root-servers.net . ns > roothints
```

This will create a new file.

You can also run

```
dig @a.root-servers.net . SOA
```

You can do this periodically to see if the SOA version number (see below) has changed.

Zone files

The root zone knows about all top-level domains directly under it, e.g., the edu and org domains as well as the country-specific domains like uk and nl.

If the example.org domain (used here for illustration purposes) were a real domain, it would not be handled by the root name servers. Instead, the nameservers for the org domain would know all about it. This is called *delegation*: the root name servers have *delegated* authority for zones under org to the name servers for the org zone. Doing delegation yourself will be explained later in the section called "Delegating a DNS zone".

A zone file is read from the named.conf file with a zone statement like

```
zone "example.org" IN {
   type master;
   file "/etc/bind/exampleorg.zone";
};
```

This is an example zone file for the zone example.org.

```
$TTL 86400
@     IN  SOA lion.example.org. dnsmaster.lion.example.org. (
         2001110700   ; Ser: yyyymmhhee (ee: ser/day start 00)
            28800    ; Refresh
             3600    ; Retry
           604800    ; Expiration
             86400 )  ; Negative caching
      IN  NS      lion.example.org.
      IN  NS      cat.example.org.

      IN  MX   0   lion.example.org.
      IN  MX  10   cat.example.org.

lion  IN   A      224.123.240.1
      IN  MX   0   lion.example.org.
      IN  MX  10   cat.example.org.

doggy IN   A      224.123.240.2
cat   IN   A      224.123.240.3

; our laptop
bird  IN   A      224.123.240.4
```

Let's examine this file in detail.

The $TTL statement

The $TTL is the *default Time To Live* for the zone. When a name server requests information about this zone, it also gets the TTL. After the TTL is expired, it should renew the data in the cache.

```
$TTL 3h
```

This sets the default TTL to 3 hours, and

```
$TTL 86400
```

this one to 86400 seconds (same as 1d (1 day)).

Note

Since BIND version 8.2 each zone file should start with a default TTL. A default value is substituted and a warning generated if the TTL is omitted. The change is defined in RFC2308.

At the same time, the last SOA time field changed meaning. Now it is the negative caching value, which is explained below.

The SOA resource record

The acronym SOA means *Start Of Authority*. It tells the outside world that this name server is recognized as the authoritative name server to query about this domain. This has two aspects: first, the parent zone org has *delegated* (granted) the use of the example.org domain to us, the second is that we claim to be the authority over this zone.

Earlier we saw the following SOA record:

```
@  IN  SOA lion.example.org. dnsmaster.lion.example.org. (
       2001110700   ; Ser: yyyymmhhee (ee: ser/day start 00)
          28800    ; Refresh
           3600    ; Retry
         604800    ; Expiration
          86400 )  ; Negative caching
```

Elements of these SOA lines are

@

the current origin, which expands to example.org (see the named.conf file).

IN

the Internet data class. Don't ask.

SOA

start of authority - that's what this section is about

lion.example.org.

the name of the machine that has the master (see the section called "Master and

slave servers") name server for this domain.

dnsmaster.lion.example.org.

The email address of the person to mail in case of trouble, with the commercial at symbol (@) normally in the email address replaced by a dot. Uncovering the reason for this is left as an exercise for the reader (something with current origin .....?)

(*five numbers*)

- The first number is the serial number. For a zone definition that never changes (e.g., the localhost zone) a single 1 is enough.

  For zones that do change, however, another format is rather common: *yyyymmddee*. That is, 4 digits for the year, two for the month, two for the day, plus two digits that start with 00 and are incremented every time something is changed. For example a serial number of

  ```
  2001110701
  ```

  corresponds to the second (!) change of the 7th of November in the year 2001. The next day, the first change will get the serial number 2001110800.

  Note

  Each time something is changed in a zone definition, the serial number must grow (by at least one). If the serial number does not grow, changes in the zone will go unnoticed.

- The second number is the refresh rate. This is how frequently a slave server (see below) should check to see whether data has been changed.

  Suggested value in rfc1537: 24h
- The third number is the retry value. This is the time a slave server must wait before it can retry after a refresh or failure, or between retries.

  Suggested value in rfc1537: 2h
- The fourth number is the expiration value. If a slave server cannot contact the master server to refresh the information, the zone information expires and the slave server will stop serving data for this zone.

  Suggested value in rfc1537: 30d
- The fifth number is the negative caching value TTL. Negative caching means that a DNS server remembers that it could not resolve a specific domain. The number is the time that this memory is kept.

  Reasonable value: 1h

The A resource record

This is the *address* record. It connects an IP address to a hostname. An example record is

```
lion   IN   A      224.123.240.1
```

This connects the name lion.example.org (remember that the current origin example.org is added to any name that does not end in a dot) to the IP address 224.123.240.1.

> **Note**
>
> Each A record has a corresponding PTR record. This is described in the section called "Reverse zone files".

The A record is used by IPv4, the current version of the IP protocol. The next generation of the protocol, IPv6, has an A6 record type. IPv6 is not discussed here.

The CNAME resource record

A CNAME record specifies another name for a host with an A record. An example of a combined A and CNAME record is

```
cat   IN  A     224.123.240.3
www   IN  CNAME cat.example.org.
```

This makes www.example.org point to cat.example.org.

The NS resource record

Specifies a name server for the zone. For example

```
@    IN SOA .....

     IN  NS    lion.example.org.
     IN  NS    cat.example.org.
```

The first thing that catches one's eye is that there is nothing before the IN tag in the NS lines. In this case, the current origin that was specified earlier (here with the SOA) is still valid as the current origin.

The name servers must be specified as *IP addresses* (can you guess why?).

Note

There must be at least two independent name servers for each domain. Independent means connected to separate networks, separate power lines, etc. See the section called "Master and slave servers" below.

The MX resource record

The MX record system is used by the mail transfer system, e.g., the sendmail daemons. The number after the MX tag is the priority. A priority of 0 is the *highest* priority. The higher the number, the lower the priority. Priority 0 will be used for the host where the mail is destined. If that host is down, another host with a lower priority will temporarily store the mail.

Example entries

```
lion   IN   A     224.123.240.1
       IN  MX   0   lion.example.org.
       IN  MX  10   cat.example.org.
```

So this example specifies that mail for lion.example.org is first sent to lion.example.org. If that host is down, cat.example.org is used instead.

To distribute mail equally among two hosts, give them the same priority. That is, if another host with priority 10 is added, they will both receive a share of mail when lion.example.org is down.

MXing a domain

Mail can be delivered to a host, as was shown in the previous section. But the Mail Transfer Agent (MTA) and the DNS can be configured in such a way that a host accepts mail for a domain. See the section called "Using Sendmail (2.206.2)" for more information about the sendmail MTA.

Considering our example, host lion.example.org can be configured to accept mail for example.org.

To implement this, place MX records for example.org, e.g.:

```
; accept mail for the domain too
example.org.  IN  MX   0   lion.example.org.
example.org.  IN  MX  10   cat.example.org.
```

in the example.org zone file.

Mail addressed to example.org will only be accepted by the MTA on lion.example.org host if the MTA is configured for this. See the section called "Using Sendmail (2.206.2)".

Reverse zone files

Each IP range has a *reverse zone*, that consists of part of the IP numbers in reverse order, plus in-addr.arpa. This system is used amongst others, to check if a hostname belongs to a specified address.

Our example.org domain uses the IP range 224.123.240.*x*. The corresponding *reverse zone* is called 240.123.224.in-addr.arpa. In named.conf this could be the corresponding entry:

```
zone "240.123.224.in-addr.arpa" IN {
   type master;
   file "/etc/bind/exampleorg.rev";
};
```

An example /etc/bind/exampleorg.rev (corresponding to the example.org domain we saw earlier) is:

```
$TTL 86400
@     IN  SOA lion.example.org. dnsmaster.lion.example.org. (
         2001110700    ; Ser: yyyymmhhee (ee: ser/day start 00)
            28800    ; Refresh
             3600    ; Retry
           604800    ; Expiration
            86400 )  ; Negative caching
      IN  NS      lion.example.org.
      IN  NS      cat.example.org.

1     IN  PTR     lion.example.org.
2     IN  PTR     doggy.example.org.
3     IN  PTR     cat.example.org.
4     IN  PTR     bird.example.org.
```

The current origin is 240.123.224.in-addr.arpa, so the entry for bird actually is

```
4.240.123.224.in-addr.arpa IN PTR bird.example.org.
```

Note

What happens if you omit the dot after example.org in this line?

The PTR record

The PTR record connects the reverse name (4.240.123.224.in-addr.arpa) to the name given by the A record (bird.example.org).

Master and slave servers

Each zone (except the ones local to the machine, such as localhost) must have at least one *master* name server. It can be supported by one or more *slave* name servers.

There should be two independent name servers for each zone. Independent means connected to a different network and other power supplies. If one power plant or network fails the resolving names from the zone must still be possible. A good solution for this is one master and one slave name server at different locations.

Both *master* and *slave* name servers are *authoritative* for the zone (if the zone was delegated properly, see the section called "Delegating a DNS zone"). That is, they both give the same answers about the zone.

The data of a zone originate on a *master* name server for that zone. The *slave* name server copies the data from the master. *Other* name servers can contact either a master or a slave to resolve a name.

This implies that the configuration must handle

- slave name server access control on the master
- other name server access control on both master and slave name servers

Configuring a master

A zone definition is defined as *master* by using the

```
type master;
```

statement inside a zone definition. See, for example, this zone statement (in named.conf)

```
zone "example.org" IN {
    type master;
    file "/etc/bind/exampleorg.zone";
};
```

defines the example.org master. Of course, The exampleorg.zone file must, of course, be created as discussed earlier.

There can be multiple independent master name servers for the same zone.

A notify statement controls whether or not the master sends DNS NOTIFY messages upon change of a master zone. The notify statement can be put in a zone statement as well as in an options statement. If one is present in both the zone and options statements, the former takes precedence. When a slave receives such a NOTIFY request (and supports it), it initiates a zone transfer to the master immediately to update the zone data. The default is notify yes;. To turn it off (when, e.g., a zone is served by masters only) set notify no;.

How does the master know which slave servers serve the same zone? It inspects the NS records defined for the zone. Extra slave servers can be specified with the also-notify statement (see the named.conf(5) manpage).

Note

Older versions of BIND used the term *primary* instead of *master*.

Configuring a slave

A zone definition is defined as *slave* by using the

type slave;

statement inside a zone definition, accompanied by the IP address(es) of the master(s). Here, for example, is the zone statement (in named.conf), which defines a example.org slave:

```
zone "example.org" IN {
    type slave;
    masters { 224.123.240.1; }; // lion
    file "db.example.org";
};
```

The file is created by the slave name server itself. The slave has no data for example.org. Instead, a slave receives its data from a master name server and stores it in the specified file.

Note that the filename has no directory component. Hence it will be written in the BIND working directory given by the directory option in named.conf. For instance, /var/cache/bind or /var/named. The name server must have write access to the file.

Note

Older versions of BIND used the term *secondary* instead of *slave*.

A stub name server

From the named.conf(5) manpage: A stub zone is like a slave zone, except that it replicates only the NS records of a master zone instead of the entire zone.

Creating subdomains

There are two ways to create a subdomain: inside a zone or as a delegated zone.

The first is to put a subdomain inside a normal zone file. The subdomain will not have its own SOA and NS records. This method is not recommended - it is harder to maintain and may produce administrative problems, such as signing a zone.

The other method is to *delegate* the subdomain to a separate zone. It is described next.

Delegating a DNS zone

A real, independent subdomain can be created by configuring the subdomain as an independent zone (having its own SOA and NS records) and delegating that domain from the parent domain.

A zone will only be *authoritative* if the parent zone has delegated its authority to the zone. For example, the example.org domain was delegated by the org domain.

Likewise, the example.org domain could delegate authority to an independent scripts.example.org domain. The latter will be independent of the former and have its own SOA and NS records.

Let's look at an example file of the scripts.example.org zone:

```
$ORIGIN scripts.example.org.
ctl  IN  A  224.123.240.16
     IN MX   0 ctl
     IN MX  10 lion.example.org.
www  IN CNAME ctl
perl IN  A  224.123.240.17
     IN MX   0 perl
     IN MX  10 ctl
     IN MX  20 lion.example.org.
```

```
bash IN  A  224.123.240.18
    IN MX  0 bash
    IN MX  10 ctl
    IN MX  20 lion.example.org.
sh   IN CNAME bash
```

Nothing new, just a complete zone file.

But, to get it authoritative, the parent domain, which is example.org in this case, must *delegate* its authority to the scripts.example.org zone. This is the way to delegate in the example.org zone file:

```
scripts  2d IN NS ctl.scripts.example.org.
        2d IN NS bash.scripts.example.org.
ctl.scripts.example.org.  2d IN  A 224.123.240.16
bash.scripts.example.org. 2d IN  A 224.123.240.18
```

That's all!

The NS records for scripts do the actual delegation. The A records *must* be present, otherwise the name servers of scripts cannot be located.

Note

It is advised to insert a TTL field (like the 2d in the example).

DNS Utilities

Three DNS utilities can help to resolve names and even debug a DNS zone. They are called dig, host and nslookup. All three are part of the BIND source distribution.

The dig program

The dig command lets you resolve names in a way that is close to the setup of a zone file. For instance, you can do

```
dig bird.example.org A
```

This will do a lookup of the A record of bird.example.org. Part of the output looks like this:

```
;; ANSWER SECTION:
bird.example.org. 1D IN A 224.123.240.4
```

```
;; AUTHORITY SECTION:
example.org.      1D IN NS  lion.example.org.

;; ADDITIONAL SECTION:
lion.example.org.  1D IN A 224.123.240.1
```

If dig shows a SOA record instead of the requested A record, then the domain is ok, but the requested host does not exist.

It is even possible to query another name server instead of the one(s) specified in /etc/resolv.conf:

```
dig @cat.example.org bird.example.org A
```

This will query name server cat.example.org for the A record of host bird. The name server that was contacted for the query will be listed in the tail of the output.

The dig can be used to test or debug your reverse zone definitions. For instance,

```
dig 4.240.123.224.in-addr.arpa PTR
```

will test the reverse entry for the lion host. You should expect something like this:

```
;; ANSWER SECTION:
4.240.123.224.in-addr.arpa.  1D IN PTR lion.example.org.

;; AUTHORITY SECTION:
240.123.224.in-addr.arpa.  1D IN NS  lion.example.org.

;; ADDITIONAL SECTION:
lion.example.org.      1D IN A       224.123.240.4
```

If you get something like

```
;; ANSWER SECTION:
4.240.123.224.in-addr.arpa. 1D IN PTR lion.example.org.240.123.224.in-addr.arpa.
```

you've made an *error* in your zone file. Given a *current origin* of 240.123.224.in-addr.arpa., consider the line:

```
4  IN PTR lion.example.org ; WRONG!
```

The dot at the end was omitted, so the current origin is appended automatically. To correct this, add the trailing dot:

```
4  IN PTR lion.example.org. ; RIGHT!
```

Note

When specifying a hostname like bird.example.org or 4.240.123.224.in-addr.arpa to **dig**, a trailing dot may be added.

The host program

The host program reports resolver information in a simple format.

When a hostname is specified as a parameter, the corresponding A record will be shown.

```
host bird.example.org
```

will result in output like

```
bird.example.org    A    224.123.240.4
```

The host program is especially useful when a hostname is wanted and an IP address is given. For example:

```
host 224.123.240.4
```

from our example hosts shows output like:

```
Name: bird.example.org
Address: 224.123.240.4
```

The following command is also possible:

```
host 4.240.123.224.in-addr.arpa
```

resolves the PTR record:

```
4.240.123.224.in-addr.arpa PTR bird.example.org
```

Note

As is the case with dig, when specifying a hostname like bird.example.org or 4.240.123.224.in-addr.arpa to host, a trailing dot may be added.

The nslookup program

The **nslookup** program is yet another way to resolve names. However, its setup is quite different. It's a tool that can be used interactively and more request types can be handled than with **dig** or **host**.

For instance, start the interactive mode by entering **nslookup** and typing:

```
ls -d example.org.
```

In result, the output will be shown in a zonefile-like format (beginning shown):

```
[lion.example.org]
$ORIGIN example.org.
@   1D IN SOA lion postmaster.lion (
              2001110800   ; serial
                   8H   ; refresh
                   1H   ; retry
                   1W   ; expiry
                   1D ) ; minimum

                 1D IN NS   lion
                 1D IN MX 0 lion
```

The first line, in square brackets, contains the name of the name server that sent the answer.

Note

The example shown requires a *zone transfer* from the connected name server. If this name server refuses zone transfers (as will be discussed below), you will of course not see this output.

A lot of commands can be given to **nslookup** in interactive mode: the help command will present a list.

Chapter 7. DNS (2.207) Home Securing a DNS Server (2.207.3)

Securing a DNS Server (2.207.3)

Revision: $Revision: 1.6 $ ($Date: 2004/01/29 20:32:13 $)

Resources and further reading: Albitz01, BINDfaq, DNShowto, Lindgreen01, Liu00, Nijssen99, Raftery01.

LPIC 2 objective 207.3

The candidate should be able to configure BIND to run as a non-root user and configure BIND to run in a chroot jail. This objective includes configuring DNSSEC statements such as key and trusted-keys to prevent domain spoofing. Also included is the ability to configure a split DNS configuration using the forwarders statement and specifying a non-standard version-number string in response to queries.

Key files, terms and utilities include:

SysV init files or rc.local
/etc/named.conf
/etc/passwd
dnskeygen

DNS Security Strategies

There are several strategies possible to make DNS more secure.

When a security bug is discovered in the BIND name server, it will in most cases be fixed within a few hours, resulting in a new BIND release. This means that some of the BIND versions that are a little older may have known (possibly security-related) bugs. Keep an eye on the BIND source site (http://www.isc.org/products/BIND/) periodically.

> **Note**
>
> Currently, BIND versions 8 and 9 are developed in parallel. Use either the newest BIND 8 or the newest BIND 9.

Consider subscribing to a security announcement list of a Linux distribution. For instance, the Debian debian-security-announce list is fast way to learn about a new security bug. See http://

lists.debian.org to subscribe.

Making information harder to get

There are ways to limit information that a normal user can query from a name server without influencing the normal name resolving. For instance, the version number can be hidden. Or access can be limited, e.g., zone transfers can be limited to a specific set of slave name servers. Both will be discussed next.

It is important to remember that hiding information is *security through obscurity*. That is, the easy way to get information is blocked, but there may be other ways to get the same information by trying several other ways. *Security through obscurity* makes it harder to get the information, but it does not make it impossible.

Hiding the version number

Since older BIND versions may have known security bugs, the BIND version number can be of great help to someone who wishes to compromise your system. The command

```
dig @target chaos version.bind txt
```

will show the version of the BIND name server on host *target*.

The BIND version can be hidden by entering a `version` statement inside the `options` statement in `named.conf`. In the following example, the version will be set to the string `hidden`.

```
options {
   // ...

   // hide bind version
   version "hidden";
};
```

The CERT article (Nijssen99) shows a way to limit access to the `bind` zone. This an alternative to replacing the BIND version.

Limiting access

There are several ways to limit access to the name server. First, an *access control list* must be defined. This is done with the `acl` statement.

```
acl "trusted" {
   localhost;
   192.168.1.0/24;
```

```
};
```

This acl defines an *access control list* with label trusted. This label is a name that can be used later, as will be shown.

Two types of connections between name servers and between name servers and resolvers are normal queries and zone transfers.

normal queries

When a host sends a *query* to a name server, it asks a name server for specific information.

The allow-query statement controls from which hosts queries are accepted.

zone transfers

A zone transfer happens when a name server sends all it knows about a zone to another name server or a resolver. Zone transfers are intended for slaves that need to get information from a master of the same zone.

Zone transfers are controlled by the allow-transfer statement.

The allow-query and allow-transfer statements can to be used inside a zone statement or inside an options statement. It can contain either an acl label (like trusted), or none or one or more IP addresses or IP ranges.

Limiting zone transfers

Both dig and nslookup can initiate a *zone transfer*. By default, both master and slave servers can give out all zone information. With dig,

```
dig @target example.org axfr
```

shows the complete information about example.org. With nslookup in interactive mode,

```
ls -d example.org
```

shows the same information.

Strictly speaking, only a slave name server needs a *zone transfer* to get the full information about the zone from a master name server for the zone.

A setup where only these zone transfers are allowed, but all other zone transfers are prohibited is described here. It consists of extras in the named.conf's of both the master and all

the slaves for a particular zone.

On master name servers. On a master name server for the example.org zone, the allow-transfer statement is used to limit zone transfers to a list of known slave servers.

```
acl "my_slave_servers" {
   224.123.240.3;  // cat.example.org
};

// ...

zone "example.org" IN {
   type master;
   // ....

   allow-transfer {
      my_slave_servers;
   };
};
```

Now only the slaves (only cat here) can request a *zone transfer* from this master name server.

On slave name servers. On a *slave* name server for the same zone, the complete zone should not be exposed at all. This too is implemented using an allow-transfer statement.

```
zone "example.org" IN {
   type slave;
   // ....

   allow-transfer {
      none;
   };
};
```

Now this slave will not accept *zone transfer* requests.

> **Note**
>
> Don't forget to do the same for the *reverse zone*.

Limiting queries

Normal queries can also be limited to a set of clients. This limits access to a certain set of hosts. For security reasons, ...

```
acl "myhosts" {
   224.123.240.0/24;
};

// ...

zone "example.org" IN {
   // ...

   allow-queries {
      myhosts;
   };
};
```

This limits queries to hosts with an IP address that starts with 224.123.240.

Controlling requests

Apart from hiding information by making it harder to get, one can control some automatic behavior of a name server that can turn out to be dangerous.

When a (malicious) name server receives a query it can respond with deliberately false data which infects the cache of the name server that sent the query. This is called *spoofing*. Although *spoofing* is not as easy as it may seem, it is wise to take precautions. Many possible spoofing methods have been made impossible in the software. In the BIND configuration steps can be taken to enhance spoofing security. For instance, *glue* can be disabled.

Turning off glue

When name server lion gets a request for, say, www.linuxdoc.org, it will query the name servers of the linuxdoc.org domain. The answer from a name server in the linuxdoc.org domain will, in most cases, contain the *names* of their name servers.

It is possible, however, that the linuxdoc.org name servers did not send the A records of their name servers in their answers. By default, our lion name server will then send request for the A records of linuxdoc.org's name servers.

This is called *glue fetching* (Liu00). Glue fetching is subject to *spoofing*, so it can be turned off.

```
options {
   // ...

   fetch-glue no;
```

```
};
```

The fetch-glue option is valid in BIND 8 but not in BIND 9. BIND 9 does not fetch glue.

Limiting effects of an intrusion

Even if you're prepared to implement any DNS fix as soon as it appears, you may be too late - even by only a couple of hours: your system could be compromised anyway. You can prepare for this by minimizing the effects of possible compromises.

Running BIND with less privileges

By default, at least in some distributions, BIND runs as root. If BIND is compromised, the attacker may get root access. This can be prevented by running BIND under a different user and group.

It is tempting to use user nobody and group nogroup for this. However, with many services running as nobody and nogroup another security issue arises when these services are able to communicate in one way or another.

The suggested way is to create a special user, e.g. named, and a corresponding group (which could also be called named), and run the name server under this user/group combination.

To use the special user and group, specify the -u and -g options to named:

```
named -u named -g named
```

On a Debian system, for instance, the start line in /etc/init.d/bind becomes

```
start-stop-daemon ... --exec /usr/sbin/named -- -u named -g named
```

(for layout reasons some options have been replaced by dots). The extra -- tells the start-stop-daemon to stop eating options, so that the named program gets these.

On a Red Hat (or compatible) system, the bind file will probably be /etc/rc.d/init.d/dns. The corresponding line to start the name server will become something like

```
daemon /sbin/named -u named -g named
```

> **Note**
>
> Make sure the name server has write access in the directory specified by the directory option in named.conf.

Running BIND in a *chroot jail*

Another way to prevent damage from a compromised name-server process is to put the process in a *chroot jail*. In such an environment, a specified directory (e.g., /var/cache/bind) will be the / (root) directory for this process. The process has no means of looking above this new root, i.e., it cannot see the rest of the filesystem anymore.

Preparing a chroot jail

Since BIND uses all sorts of files, these files must be copied to a directory structure below the new root that mimics the original directory structure. Article Lugo00 lists files to copy. Here is an abstract:

- You will need the etc, dev, lib, sbin or usr/sbin, and var/run directories under the new root.
- You'll probably need the /dev/null device under the new root:

  ```
  mknod -m 666 /var/cache/bind/dev/null c 1 3
  ```

  will create it.
- You will need a passwd and a group file in the new /etc:

  ```
  cd /var/cache/bind/etc
  cp -p /etc/{passwd,group} .
  ```

  This copies the literal passwd and group files. An alternative is to create special passwd and group files, as will be shown in the section called "Combining special user and chroot".

  You may also need to copy in some other stuff.

  ```
  cd /var/cache/bind/etc
  cp /etc/ld.so.cache .
  cp /etc/localtime .
  ```

- The configuration must also be present under the new root. The chrooted named will need to find it when it receives a reload request. The configuration can be placed in /var/cache/bind/etc/bind.

  Note that each directory specified in the new named.conf - the ones specified with the directory option - is relative to the new root directory.
- The new lib directory must contain at least all the shared libraries used by bind. For instance, on my system,

  ```
  ldd /usr/sbin/named
  ```

yields

```
libc.so.6 => /lib/libc.so.6 (0x40015000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

When looking in (real) /lib, libc.so.6 is a symlink to libc-2.1.3.so and ld-linux.so.2 is a symlink to ld-2.1.3.so. Both the symlinks and the files they point to must be copied to the lib directory under the new root:

```
cd /var/cache/bind/lib
cp -pd /lib/{libc.so.6,libc-2.1.3.so,ld-linux.so.2,ld-2.1.3.so} .
```

- Both the named and the named-xfer programs must be present under the new root. The ndc program may be of help too. On my system, they are in /usr/sbin:

```
cp -p /usr/sbin/named{,-xfer} /var/cache/bind/usr/sbin
cp -p /usr/sbin/ndc /var/cache/bind/usr/sbin
```

On my system, it is in /usr/sbin and uses the same shared libraries as named.

Running BIND chrooted

To get the BIND name server in a *chroot jail*, simply add the -t and a directory name to the command line used to start the name server.

Note

It is also possible to use the chroot command to start the name server. This has exactly the same effect as using the -t option, so it is not covered here.

So, for example, in /etc/init.d/bind:

```
start-stop-daemon ... --exec /usr/sbin/named -- -t /var/cache/bind
```

This will make /var/cache/bind the new root directory for the name server.

Or, in /etc/rc.d/init.d/dns:

```
daemon ... /sbin/named /var/cache/bind
```

This makes /var/cache/bind the new root directory for the name server.

Remember that the double dash -- after named tells the start-stop-daemon to pass options after the double dash to named.

Important

The BIND name server activates the chroot immediately after all command lines are read. That is, before the configuration files are read.

Configuration for a chrooted BIND

The configuration files are read *after* the name server has chrooted to the specified directory. As a consequence, every configuration file (e.g. zone files and the named.conf file) must be present in a configuration directory (e.g., /etc/bind) under the chroot directory.

One of the things that must be reconfigured is logging. The normal logging to syslog will not work because of the chroot jail (cannot reach the Unix socket /dev/log and cannot recreate it). Here is an example logging statement from the named.conf file inside the chroot environment that can solve this problem:

```
logging {
   channel some_log {
      file "bind.log" versions 3;
      severity info;
   };

   category default { some_log; };

   // ...
};
```

This will write logging information to the file /var/cache/bind/var/cache/bind/bind.log, which was composed as follows:

/var/cache/bind (the first)

the chroot directory specified by the -t option when starting named

/var/cache/bind (the second)

the directory specified by the directory statement inside the options statement

bind.log

the name of the log file specified by the logging statement

Zone files, if any, are also placed inside the chroot environment. For instance, if named.conf contains a definition of a master zone like

```
zone "example.com" IN {
```

```
    type master;
    file "/etc/bind/example.com.zone";
};
```

the example.com.zone file will be in /var/cache/bin/etc/bind.

Combining special user and chroot

Using a combination of running the name server under a special user and group and a chroot environment will yield even better security. If the zone- and configuration files are unwritable by the special user and group the name server is running under, they cannot be compromised by a name server break in.

> **Note**
>
> Inspecting the source for BIND 8.2.5 reveals the order in which things happen:
>
> - parsing of commandline options
> - chrooting
> - go in the background and write pid file
> - change to new user and group
>
> This means that the directory where the pid file is written (e.g., /var/run) needs write permission for root, but the new caching directory needs write permission for the specified user and group.

The chroot environment creates the possibly of special /etc/passwd and /etc/group files, that are completely different from the ones in the *real* /etc. The special user and group used by the name server may not even exist in the real /etc/passwd. This is the way to create the special files

```
cd /var/cache/bind/etc
echo "named::22" > group
echo "named:x:22:22:named:/var/cache/bind:/bin/false" > passwd
```

However, the named program needs to know the special uid/gid's at startup. So you must also do:

```
echo "named::22" >> /etc/group
echo "named:x:22:22:named:/var/cache/bind:/bin/false" >> /etc/passwd
```

Do not forget to add something like

```
named:!:11222:0:99999:7:::
```

This must be added to both /etc/shadow files (one in the real /etc, and one relative to the chroot environment) if shadow passwords are active.

Securing name server connections

If a malicious name server manages to infect zone transfers, it can poison the zone information on a slave nameserver. Zone transfers can be limited to certain hosts by the allow-transfer statement in named.conf.

Queries can also be poisoned. Queries can also be limited to dedicated hosts or host groups using the allow-query statement.

Limiting access to certain hosts may solve the risk for poisoning if that hosts can be trusted. But how can you be sure of this? If a name server receives an answer, it simply cannot be sure that the name server that sent the answer really is the name server it claims to be.

Signing an answer by the name server that sends it may solve this. Now the name server that receives the answer can verify the signature. And, it knows whether or not the answer really originated on the name server that the sender claims to be.

According to the lpic2 objective you must be able to use BIND 8's **dnskeygen**. This helps you to configure secure transactions between name servers.

> **Note**
>
> The **dnskeygen** command (which will be called **dnssec-keygen** in BIND version 9) is part of the DNSSEC security extension for DNS (fully described in Albitz01).
>
> DNSSEC is not really useful in BIND version 8 due to a limited implementation. It is fully implemented in BIND version 9 which is described in the Albitz and Liu book.
>
> DNSSEC was in development for several years, but did not catch on in the market (the fact that it will be fully implemented in BIND 9 may change this). Moreover, according to Lindgreen01, nowadays most known ways to compromise a DNS server are handled either by software or can be prevented by strict configuration. Therefore, the author states, it may be better to stop developing DNSSEC in favor of a new design of the domain name system.

Using the dnskeygen command

The **dnskeygen** generates public, private and shared keys for DNS, according to the manual page. Dnskeygen (DNS Key Generator) is a tool to generate and maintain keys for DNS Security within the domain name system. The **dnskeygen** command can generate public and private keys to authenticate zone data and shared secret keys to be used for Request/ Transaction signatures.

First choose the type of key:

a *zone key* (specify -z)

used to sign zone data

a *host key* (specify -h)

used to sign host data

a *user key* (specify -u)

Strangely enough, **dnskeygen** can also generate a key applied by a user, for example for signing email.

Note

Exactly one of the -z, -h or -u flags must be specified.

Next, choose the algorithm. An algorithm is selected with a flag followed by a number which indicates the key size.

DSA/DSS (specify -D)

generate a DSA/DSS key.

HMAC-MD5 (specify -H)

generate a HMAC-MD5 key.

RSA (specify -R)

generate an RSA key. If the -F flag is also specified (after -R and the size) the program uses a large exponent for the generation of the RSA key.

Note

Exactly one of the -D, -H or -R flags must be specified.

By default, a generated key can be used for both authentication and encryption. Two flags change this:

-a

the key *cannot* be used for authentication

-c

the key *cannot* be used for encryption

The last two arguments are -n followed by the name of the key. Both are required.

Note

This does not cover the use of **dnskeygen** in detail. See the manual page.

Generated key files

The **dnskeygen** creates two files: K*name*+*algorithm*+*footprint*.private and K*name*+*algorithm*+*footprint*.key.

Suppose the command given is

```
dnskeygen -H 768 -h key.example.com.
```

Now two output files are created called Kkey.example.com.+157+12345.private and Kkey.example.com.+157+12345.key.

The file Kkey.example.com.+157+12345.private looks like

```
Private-key-format: v1.2
Algorithm: 157 (HMAC)
Key: 5VBiSy...
```

The file Kkey.example.com.+157+12345.key has the following contents

```
key.example.com. IN KEY 513 3 157 5VBiSy...
```

The generated keys (abbreviated in both cases) are identical in both files.

Using the key

To use the generated key put a key statement in named.conf. This looks like

```
key key.example.com. {
   algorithm "hmac-md5";
   secret "5VBiSy...";
};
```

This requires, of course, that named.conf be unreadable for anyone but the name server. The key can also be put in a separate file that can be included with the include statement, so that

the included file is the one that has limited readability.

Next, define the server that connects using the key. On server 224.123.400.2 define that host 224.123.400.1 can use this key:

```
server 224.123.400.1 {
    keys key.example.com.;
};
```

On name server 224.123.400.1 do the same for server 224.123.400.2.

Now transfers (for instance) can be limited to those zones that are signed by this key:

```
zone "example.com" {
   type master;
   file "example.com.zone";
   allow-transfer { key key.example.com.; };
};
```

Internal DNS

Suppose your division is located outside the firm's main building. The workgroup will have its own name servers. Internally, hosts will be members of the exworks (short for *exampleworks*) domain, that is, directly under the root (.) domain. IP addresses of all hosts begin with 192.168.72, so the reverse zone 72.168.192.in-addr.arpa will also be maintained.

The problem with this is that the root name servers don't know anything about the exworks and 72.168.192.in-addr.arpa zones. Nor should they: there should be no request concerning these internal zones to one of the root name servers. Since the DNS system was not designed for use behind a firewall, some tricks must be applied. A poor-man's solution is to try to limit any traffic from the internal name server to the root servers. But, using a split-level DNS setup is likely to be a much better solution. Two split-level DNS setups will be shown.

Limiting negotiations

BIND behind a dial-up connection can be a nuisance: every time a DNS wants to communicate with a root name server, an automatic dialer sets up a connection. You stop BIND from doing this by putting

```
// prevent dialup access
heartbeat-interval 0;
dialup yes;
```

inside the options statement of the named.conf file.

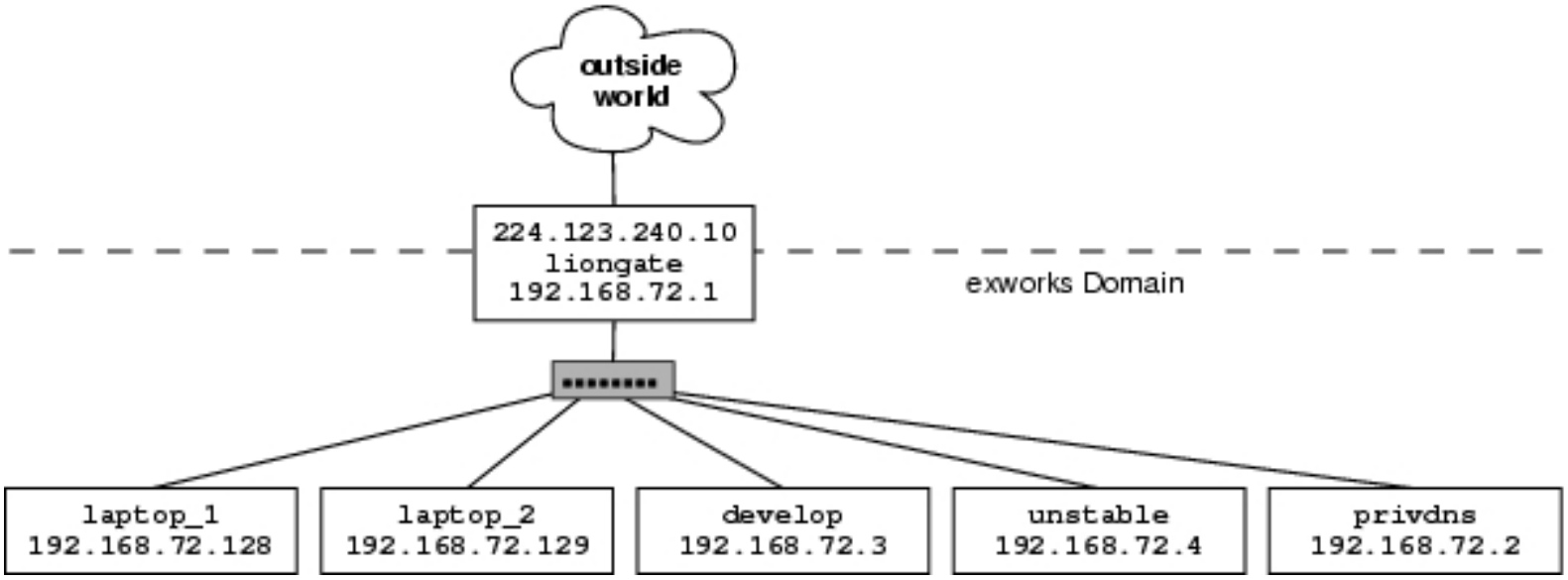
This trick can also be used to limit communication from an internal zone like exworks. The exworks and 72.168.192.in-addr.arpa zones are implemented as conventional master zones as described earlier in this chapter.

The fundamental problem with this is that the real root servers still don't delegate the exworks (and corresponding reverse) domain. A better solution therefore is to put the internal zones together with the root name server definition into a separate DNS implementation. This requires at least two independent name servers, and they can be set up in two different ways, as will be described below.

Split DNS: stand-alone internal master

The first approach consists of a stand-alone name server (that is, master for the internal zones) and another name server on another host that can be used to resolve names from both the outside world and the internal zones.

The figure below presents the exworks domain that is used as an example.



The exworks network.

The two name servers in this example will be on different hosts. The first name server runs on privdns and will provide internal zone information. The other name server is on liongate, which will be a forwarding name server for both the internal zones and the outside world.

The host privdns (for *private DNS*) will contain the DNS for both the exworks and 72.168.192.in-addr.arpa zones. This name server will have the following characteristics:

- it will be master for the root domain, but no DNS except itself will be able to access this information
- it will be master for the the exworks and 72.168.192.in-addr.arpa domains. Only other name

servers inside the building will be able to access this information

On the other hand, liongate will do the following:

- do forwarding for the internal zones (this should be done *first*)
- do forwarding for the outside world

Configuring the master on privdns

First, prepare a master file for the internal root zone For example:

```
$TTL 25h
.       IN  SOA privdns.exworks. postmaster.privdns.exworks. (
           2001121000    ; yyyymmhhee (ee == ser/day start 00)
               28800
                3600
              604800
               86400 )
        IN  NS privdns.exworks.
privdns.exworks. IN  A  192.168.72.2

; glue records
exworks.              IN NS privdns.exworks.
72.168.192.in-addr.arpa. IN NS privdns.exworks.
```

Note the glue records that delegate the exworks and 72.168.192.in-addr.arpa zones.

Next, add a zone statement in named.conf that points to the master file for the root zone:

```
// ROOT MASTER zone for internal DNS server
zone "." IN {
        type master;
        file "/etc/bind/internal.root.zone";
        allow-transfer { none; };
        allow-query    { none; };
};
```

Using the type master tells the DNS that this really is a master server for the root zone. The root zone should not be known to any other name servers, therefore the allow-transfer and allow-query are set to none.

Note

The root zone definition type hint, as present in a default caching-only server,

should be turned *off* (by not including it in the named.conf file.

Now, prepare zone files for the exworks and 72.168.192.in-addr.arpa zones, and add the corresponding zone entries in the named.conf file. The entry for exworks should look like this:

```
zone "exworks" IN {
        type master;
        file "/etc/bind/exworks.zone";
        allow-transfer { none; };
        allow-query    { 192.168.72.1; }; // liongate
};
```

The name server on liongate is the one providing names to the other hosts, so the IP address of liongate must be listed in the allow-query field.

The same must be done for the corresponding reverse zone.

Make sure the options statement contains

```
recursion no;
fetch-glue no;
```

These statements tell the name server it should not accept queries for zones other than the ones it knows about.

Hosts in the exworks zone should point their resolv.conf to liongate. That is, the file should contain the line:

```
nameserver 192.168.72.1
```

where 192.168.72.1 is the IP address of liongate. On liongate itself, however,

```
nameserver 127.0.0.1
```

is more efficient.

Note

The entries in resolv.conf on privdns should not point to the name server on privdns itself. If the host is to be used for purposes that require name resolving, it is better to point the entries to the name server on liongate.

Configuring DNS on liongate

The functionality of the DNS is twofold: first it should resolve names of the exworks zone (and corresponding reverse zone), secondly, it should resolve names from the outside world.

With the following forwarders statement queries to this name server are being forwarded to one on the listed IP addresses (remember that forwarders is part of the options statement):

```
forwarders {
   192.168.72.2;   // privdns
   224.121.121.99; // ISP's DNS
};
```

The name servers are tried in the order in which they are listed. Requests are forwarded to privdns first. If that one does not have the answer, the DNS of the ISP is contacted instead.

Note that the IP address of privdns should be mentioned *first*: we don't want requests for internal names be sent to the ISP.

Alternatively, the name server on liongate can be configured as slave for the exworks domain and corresponding reverse domain. Because this is a *must* in the setup where both name servers are on one host (which is described next), it is not discussed here.

Alternatives

There are two alternatives to this setup. Both require two separate name servers.

The first alternative is to use *two* nameserver statements in resolv.conf:

```
nameserver 192.168.72.2
nameserver 192.168.72.1
```

The first IP address points to privdns, the second to liongate. In this situation, privdns should also accept queries from 192.168.72.0/24 and the forwarders statement on liongate should not contain the 192.168.72.2 (the IP address of privdns). The downside of this is that there is no single name server entry-point.

The second alternative is to use a slave setup for the exworks (plus corresponding reverse) domain. This situation is required when two name servers are running on the same host. This will be elaborated below, so it will not be discussed here. For this to work, the forwarders statement on liongate should not contain the IP address of privdns.

Split DNS: two DNS servers on one machine

Suppose the same network exists as above, but with one important difference: host privdns is *not* available. This implies that the *internal* name server must now run on one of the other

hosts, which also need access to the outside world.

In the situation described here, both name servers will be running on one host. At least one of these must run chrooted.

Two name servers on one host

On oder to do this, the following settings must be made:

- **The *internal* name server.** There is a master name server that serves the exworks and corresponding reverse domains. It runs chrooted, under its own uid/gid, listening at a special port. It will not do any other resolver work. This name server will be referred to as the *internal* name server.
- **The *visible* name server.** The other name server does not run chrooted, but it does have its own uid/gid. It serves as a slave for the exworks and 72.168.192.in-addr.arpa domains. It also forwards other requests to the ISP. This name server will be referred to as the *visible* name server, since this is the name server that will be visible to other hosts.

The nameserver line in the resolv.conf file points to 127.0.0.1; other hosts in the exworks domain point their resolver to 192.168.72.1 (the IP address of liongate).

Configuring the internal name server

The name server is put into a chroot jail with own user and group ID's. The name server will chroot to /var/cache/bindInternal. It will run as user inamed with UID 5301 and as group with the same name and GID 5301.

This chrooted name server should be able to write to some directories. To allow this, these directories should be owned by the the inamed UID and GID. The directories (inside the chroot jail) that need this are /var/run (because it needs to write named.pid) and /var/cache/bind (because it needs to write e.g. a logfile).

The configuration directory will be /var/cache/bindInternal/etc/bind, so that references from the named.conf file in that directory will be to /etc/bind. Files in that directory include the master file for the root zone and the zone files for both the exworks and 72.168.192.in-addr.arpa domains.

Here are the three master zone definitions:

```
options {
   directory "/var/cache/bind";
   listen-on port 5353 { 127.0.0.1; };
   recursion no;
   fetch-glue no;
};
```

```
// NOT SHOWN HERE, described elsewhere:
// - special chroot logging, see above
// - zones "localhost", "127.in-addr.arpa", "0.in-addr.arpa" and
//   "255.in-addr.arpa as usual

// ROOT MASTER for internal DNS server
zone "." IN {
   type master;
   file "/etc/bind/internal.root.zone";
   allow-transfer { none; };
   allow-query    { none; };
};
// NO root zone, type hint definition!

// EXWORKS ZONES
zone "exworks" IN {
   type master;
   file "/etc/bind/exworks.zone";
   allow-transfer { 127.0.0.1; };
   allow-query    { 127.0.0.1; };
};

zone "72.168.192.in-addr.arpa" IN {
   type master;
   file "/etc/bind/exworks.rev";
   allow-transfer { 127.0.0.1; };
   allow-query    { 127.0.0.1; };
};
```

The directory refers to a location inside the chrooted tree. The listen-on specifies that this name server should only listen on port 5353 on the internal address 127.0.0.1. The other two, recursion and fetch-glue, prevent resolving anything but the zones that were defined here.

The root zone is purely local, it is only for this name server. Therefore, the configuration does not allow queries and transfers.

The exworks and 72.168.192.in-addr.arpa zone definitions allow for internal zone transfers and queries, both specifically to the other name server running there (the visible one). No other transfers and queries are allowed for these zones.

Configuring the visible name server

This section will describe how to configure the *visible* name server (if in doubt read the section

called "Two name servers on one host" again). Remember it does not run chrooted and listens on a normal port. Instead, it runs under its own UID/GID.

The visible name server should be able to answer queries about both the inside and the outside worlds.

> **Note**
>
> The *visible* name server on liongate could use forwarding to connect to the internal name server if the software allowed you to specify a port. While BIND 8 does not allow this, BIND 9 will. In any case, it is easy to use a slave setup instead of forwarding.

The named.conf for the visible name server, with common parts omitted, looks like this:

```
// /etc/bind/named.conf
// bind visible configuration on liongate

options {
   directory "/var/cache/bindVisible";

   // point to the ISP's name server for outside world
   forwarders {
      224.121.121.99; // ISP's DNS
   };
};

// NOT SHOWN HERE, described elsewhere:
// - logging as usual
// - root zone type hint as usual
// - zones "localhost", "127.in-addr.arpa", "0.in-addr.arpa" and
//   "255.in-addr.arpa as usual

// point to port 5353 for these zones, be slave for both
zone "exworks" IN {
    type slave;
    masters port 5353 { 127.0.0.1; };
    file "exworks.slave";
    allow-transfer { none; };
    allow-query    { 127.0.0.1; 192.168.72.0/24; };
};

zone "72.168.192.in-addr.arpa" IN {
    type slave;
```

```
    masters port 5353 { 127.0.0.1; };
    file "72.168.192.in-addr.arpa.slave";
    allow-transfer { none; };
    allow-query    { 127.0.0.1; 192.168.72.0/24; };
};
```

This implements the following:

- a name server that performs slave resolving for the exworks and 72.168.192.in-addr.arpa zones
- forwarding to resolve names from the outside world

The directory specified by the directory statement must be writable by the running name server. That is, if the name server is running as user vnamed and group vnamed, the directory /var/cache/bindVisible must be owned by user vnamed and group vnamed. The slave files exworks.slave and 72.168.192.in-addr.arpa.slave will be put there.

Let's look at the slave zone definitions. Both refer to a master at port 5353 on the same host. Both definitions do not allow full zone transfers. The only zone transfers are from master (at port 5353) to slave (the configuration for the *master* allows this, see the section called "Configuring the master on privdns"). Normal queries are allowed for the localhost (127.0.0.1, i.e., internal connections on liongate), as well as all hosts in the exworks zone (192.168.72.*x*).

On liongate the resolv.conf file will contain

```
nameserver 127.0.0.1
```

that is, it points to the *visible* name server which listens on port 53. Other hosts in the exworks domain will have

```
nameserver 192.168.72.1
```

(the IP address of liongate) in their resolv.conf.

> **Note**
>
> It is not possible to specify a portnumber in resolv.conf. The specified name server (s) will be contacted at port 53.

A problem

There is one problem with this scheme that results in extra maintenance. Normally, a master name server sends a *notify* to a slave name server when a zone definition changes. But there is no way to tell a name server that there are two name servers listening at the same IP

addresses on different ports. In fact, the also-notify statement cannot contain port information.

There is a simple solution to this problem. After a master zone is changed, and the internal name server has reloaded the zone data, the visible name server has to be be restarted manually.

Prev | Up | Next

Create And Maintain DNS Zones (2.207.2) | Home | Chapter 8. Web Services (2.208)

Chapter 8. Web Services (2.208)

Prev Next

Chapter 8. Web Services (2.208)

Revision: $Revision: 1.4 $ ($Date: 2004/03/03 15:03:36 $)

This objective has a weight of 6 points and contains the following objectives:

Objective 2.208.1; Implementing a Web Server (2 points)

> Candidates should be able to install and configure an Apache web server. This objective includes monitoring Apache load and performance, restricting client-user access, configuring **mod_perl** and PHP support, and setting up client user authentication. Also included is the configuration of Apache server options, such as maximum requests, minimum and maximum servers, and clients.

Objective 2.208.2; Maintaining a Web Server (2 points)

> Candidates should be able to configure Apache to use virtual hosts for web-sites without dedicated IP addresses. This objective also includes creating an SSL certification for Apache and defining SSL definitions in configuration files using OpenSSL, as well as customizing file access by implementing redirect statements in the configuration files of Apache.

Objective 2.208.3; Implementing a Proxy Server (2 points)

> Candidates should be able to install and configure a proxy server using Squid. This objective includes implementing access policies, setting up authentication and configuring memory usage policies.

Implementing a Web Server (2.208.1)

Candidates should be able to install and configure an Apache web server. This objective includes monitoring Apache load and performance, restricting client-user access, configuring **mod_perl** and PHP support, and setting up client user authentication. Also included is the configuration of Apache server options, such as maximum requests, minimum and maximum servers, and clients.

Revision: $Revision: 1.11 $ ($Date: 2004/08/13 07:43:14 $)

Key files, terms and utilities include:

access.log
.htaccess
httpd.conf
mod_auth
htpasswd
htgroup

Resources: LinuxRef06; Coar00; Poet99; Wilson00; Engelschall00; PerlRef01; Krause01; the **man** pages for the various commands.

Installing the Apache web-server

As of this writing (February 2002), there are two versions of the Apache web-server available for download: a production version (version 1.3.x) and a beta version of the new version 2.0. Apache 2.0 will include several new enhancements, but is not intended for production use (yet). If you are not familiar with software development, and wish to use a stable, working, web server, you should use Apache 1.3 until at least the 2.0 series gets out of beta. In this chapter, we focus on the stable version.

Apache can be built from source, but in most cases it is already part of your distribution. The next step is to edit the configuration files for the server: by default, these files are called srm.conf, access.conf and httpd.conf. You should also have an additional file called mime.types (typically found in the /etc directory or in the same directory your configuration files are). The mime.types file usually does not need editing.

Your distribution probably contains a pre-cooked set of configuration files. Typical locations for these configuration files are /etc/httpd/ or /etc/apache/. To help you get started, most distributions (and of course the Apache source distribution too) provide examples of these configuration files, called srm.conf-dist, access.conf-dist and httpd.conf-dist. Copy or rename these files dropping off the -dist. Then edit each of the files. Read the comments in each file carefully. Failure to set up these files correctly could lead to your server not working or being insecure.

First, edit httpd.conf. This sets up the general attributes of the server: the port number, the user it runs as, etc. Next, edit the srm.conf file; this sets up the root of the document tree, sets such special functions as server-parsed HTML or internal imagemap parsing, etc. Finally, edit the access.conf file to, at least, set the base cases of access. In addition to these three files, the server behavior can be configured on a directory-by-directory basis by using .htaccess files in directories accessed by the server, provided you configured this in the httpd.conf file using the AllowOverride option).

Modularity

Apache, like many other successful open source projects, has a modular source code architecture. This means that, to add or modify functionality, you do not need to know the whole code base. You can custom build the server with only the modules you need and include your own modules.

Extending Apache can be done in C or in a variety of other languages using the appropriate modules. These modules expose Apache's internal functionality to different programming languages such as Perl or Tcl. There are many modules available, too many to list in this book. If you have any questions about the development of an Apache module, you should join the Apache-modules mailing list at *http://modules.apache.org*. Remember to do your homework first: research past messages and check all the documentation on the Apache site. Chances are good that someone has already written a module that solves the problem you are experiencing.

The modular structure of Apache's *source code* should not be confused with the functionality of *run-time loading* of Apache modules. Run-time modules are loaded after the core functionality of Apache has started and are a relatively new feature. In older versions, to use the functionality of a module, it needed to be compiled in in during the *build* phase. Current implementations of Apache are capable of loading modules *run-time*, see DSO.

Run-time loading of modules (DSO)

Most modern Unix derivatives include a mechanism called dynamic linking/loading of Dynamic Shared Objects (DSO). This provides a way to build a piece of program code in a special format for loading at run-time into the address space of an executable program. This loading can usually be done in two ways: either automatically by a system program called ld.so when an executable program is started, or manually, from within the executing program via a system interface to the Unix loader through the system calls **dlopen()**/**dlsym()**.

In the latter method the DSO's are usually called shared objects or DSO files and can be named with an arbitrary extension (although the canonical name is foo.so). These files usually are installed in a program-specific directory. The executable program manually loads the DSO at run-time into its address space via **dlopen()**.

The fact that Apache already uses a module concept to extend its functionality, and internally uses a dispatch-list-based approach to link external modules into the Apache core functionality predestined Apache for using DSO's. Starting with version 1.3 Apache began using the DSO mechanism to extend its functionality at run-time. As of then the configuration system supports two optional features for taking advantage of the modular DSO approach: compilation of the Apache core program into a DSO library for shared usage and compilation of the Apache modules into DSO files for explicit loading at run-time.

Tip

While Apache is installed by default in most Linux distributions, all versions do not support dynamic modules. To see if your version of Apache supports them,

execute the command **httpd -l** which lists the modules that have been compiled into Apache. If mod_so.c appears in the list of modules then your Apache server can use dynamic modules.

APache eXtenSion (APXS) support tool

The APXS is a new support tool from Apache 1.3 which can be used to build an Apache module as a DSO outside the Apache source-tree. It knows the platform dependent build parameters for making DSO files and provides an easy way to run the build commands with them.

Encrypted webservers: SSL

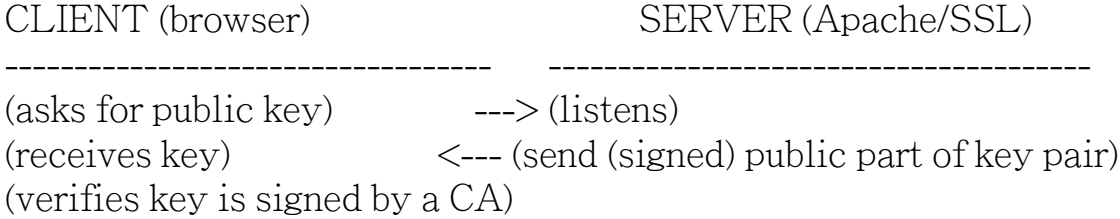
Apache has been modified to support *Secure Socket Layers* for Secure On-Line transactions. The Secure Sockets Layer protocol (SSL) is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g., TCP/IP) and the application protocol layer (e.g., HTTP). SSL provides secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity and encryption for privacy. There are a currently two versions of SSL still in use: versions 2 and 3. Additionally, there is the successor to SSL, TLS (version 1, which is based on SSL), designed by the IETF.

Public key cryptography

SSL uses Public Key Cryptography (PKC), also known as asymmetric cryptography. Public key cryptography is used in situations where the sender and receiver do not share a common secret, e.g., between browsers and web-servers, but wish to establish a trusted channel for their communication.

PKC defines an algorithm which uses two keys, each of which may be used to encrypt a message. If one key is used to encrypt a message, then the other must be used to decrypt it. This makes it possible to receive secure messages by simply publishing one key (the public key) and keeping the other secret (the private key). Anyone may encrypt a message using the public key, but only the owner of the private key will be able to read it. For example, Joan may send private messages to the owner of a key-pair (e.g., your web-server), by encrypting the messages using the public key your server publishes. Only the server will be able to decrypt it.

Figure 8.1. Public key exchange



(decrypts message with public key)
"Al;ao03lls eis oe sowlole!lsli@ " ---> (decrypts message w/private key)
---------- ----------

A secure web-server (e.g., Apache/SSL) uses HTTP over SSL, using port 443 by default (can be configured in httpd.conf). Within the browser, this is signified by the use of the https scheme in the URL. The public key is exchanged during the set-up of the communication between server and client (browser). That public key is signed (it contains a digital signature e.g., a message digest) by a so-called CA (Certificate Authority). The browser contains a number of so-called *root*-certificates: they can be used to determine the validity of the CA's that signed the key.

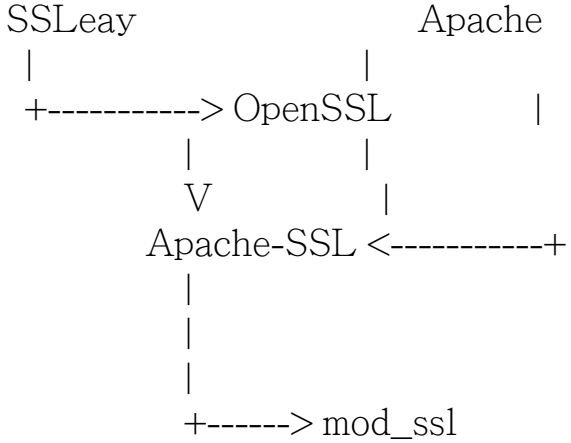
Various Apache and SSL related projects

A number of solutions are available to enable Apache to use SSL on Linux:

- commercially licensed: Raven, Stronghold
- Apache with SSLeay or Open-SSL, aka Apache-SSL
- mod_ssl

It can be quite confusing to find out the difference between SSLeay, OpenSSL, Apache/SSL and **mod_ssl**. Therefore, the relation between these components is shown in Figure 8.2, "Relations between Apache and SSL related projects".

Figure 8.2. Relations between Apache and SSL related projects



Eric A. Young and Tim J. Hudson created **SSLeay** - a library containing encryption functions. Another team, lead by Ralf Engelschall and Ben Laurie, used this library as a starting point to create a complementary set of cryptography software named **OpenSSL**. A team lead by Ben Laurie combined **OpenSSL** with the Apache webserver to create Apache-SSL. In 1998, Ralf Engelschall and his team derived **mod_ssl** from **Apache-SSL**.

mod_ssl is *not* a replacement for **Apache-SSL** - it is an alternative. It is a matter of personal choice as to which you run. **mod_ssl** is what is known as a "split" - i.e., it was originally derived from **Apache-SSL**, but has been extensively redeveloped. Many people find it very easy to install.

There are a number of commercial products available: Red Hat's Secure Web Server (which is based on **mod_ssl**), Covalent's Raven SSL Module (also based on **mod_ssl**) and C2Net's product Stronghold (based on a different evolution branch named Sioux up to Stronghold 2.x and based on **mod_ssl** since Stronghold 3.x).

Apache-SSL

To use Apache-SSL, you will need to do the following: acquire and install Apache with the proper patches, acquire and install OpenSSL, generate a key-pair and either sign the public part of it yourself, thus creating a certificate, or have it signed by a commercial Certificate Authority (CA).

The Apache-SSL distribution is simply a set of patches for Apache (available for versions 1.2.0+ and 1.3.0+), some extra source files, a few README files and a few example configuration files. The patches must be applied to the Apache source, and the result compiled and linked with OpenSSL.

Before installing it, you will need to be sure that you have unpacked the source code for the appropriate version of Apache. The modified source will still compile a standard Apache as well as Apache-SSL.

After installation of the software, you will need to configure Apache as usual. Also, a number of additional directives should be used in the Apache(-SSL) configuration files to configure the secure server for example, the location for the key-files. As it's beyond the scope of this book to document these directives we recommend reading the Apache-SSL documentation and the materials found on the *Apache-SSL website*.

Apache with mod_ssl

To use **mod_ssl** you will need to acquire and install Apache, patch it, and install and configure the module. You also need to acquire and install OpenSSL, generate a key-pair, and either sign the public part of it yourself, thus creating a certificate, or have it signed by a commercial Certificate Authority (CA).

The **mod_ssl** package consists of the SSL module itself - and, surprisingly, a set of patches for Apache itself. This may puzzle you at first: why you need to patch Apache to install the **mod_ssl** *module*? Well, the standard API that Apache uses for it's modules is unable to communicate with the SSL module. Therefore, the source patches add the Extended API (EAPI). In other words: you can only use the **mod_ssl** module when Apache's core code contains the Extended API. When building **mod_ssl**, the Apache source tree is automatically

altered for you, adding the Extended API.

After installation of the software you will need to configure Apache as with Apache-SSL. Some additional directives should be used to configure the secure server - for example the location of the key-files. It's beyond the scope of this book to document these directives, however, you can find them in the **mod_ssl** documentation and on the ***mod_ssl** web-site*.

Monitoring Apache load and performance

Apache is a general web-server and is designed to be correct first and fast second. (Even so, its performance is quite satisfactory.) Most sites have less than 10Mbits of outgoing bandwidth, which Apache can fill using only a low end Pentium-based computer. In practice, sites with more bandwidth require more than one machine to fill the bandwidth due to other constraints (such as CGI or database transaction overhead). For these reasons, the development focus has stayed on correctness and configurability.

The single biggest hardware issue affecting web-server performance is RAM. A webserver should never ever have to swap. It increases the latency of each request beyond a point that users consider "fast enough". This causes users to hit stop and reload, further increasing the load. You can, and should, control the MaxClients setting so that your server does not spawn so many children that it starts swapping.

An Open Source system that can be used to periodically load-test pages of web-servers is **Cricket**. **Cricket** can be easily set up to record page-load times, and it has a web-based grapher that will generate charts to display the data in several formats. It is based on **RRDtool**, whose ancestor is MRTG (short for "Multi-Router Traffic Grapher"). RRDtool (Round Robin Data Tool) is a package that collects data in "round robin" databases; each data file is fixed in size so that running **Cricket** does not slowly fill up your disks. The database tables are sized when created and do not grow larger over time. As the data ages, it's averaged.

Apache access_log file

The access_log gives a generic overview of the access to your web-server. The format of the access log is highly configurable. The format is specified using a format string that looks much like a C-style **printf** format string. A typical configuration for the access log might look as follows.

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

This defines the nickname common and associates it with a particular log format string. The above configuration will write log entries in a format known as the Common Log Format (CLF). This standard format can be produced by many different web servers and read by

many log analysis programs. The log file entries produced in CLF will look something like this:

127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326

and contain the following fields:

1. IP address of the client (%h)
2. RFC 1413 identity determined by **identd** (%l)
3. userid of person requesting (%u)
4. time server finished serving request (%t)
5. request line of user (%r)
6. status code servers sent to client (%s)
7. size of object returned (%b).

Restricting client user access

Apache is aware of two methods of access control:

discretionary access control (DAC)

: check the validity of the credentials given by the user, e.g. username/password (you can change these at your discretion);

mandatory access controls (MAC)

: validate aspects that the user cannot control, for example, your DNA sequence, fingerprint or retinal patterns.

In Web terms, and Apache terms in particular, discretionary controls are based on usernames and passwords, and mandatory controls are based on things like the IP address of the requesting client.

Apache uses modules to authenticate and authorise users. Generally, modules let you store the valid credential information in one or another format. The **mod_auth** module, for instance, looks in normal text files for the username and password info, and **mod_auth_dbm** looks in a DBM database for it.

Below is a list of the security-related modules that are included as part of the standard Apache distribution.

mod_auth

: This is the basis for most Apache security modules; it uses ordinary text files for the

authentication database.

mod_access

This is the only module in the standard Apache distribution which applies mandatory controls. It allows you to list hosts, domains, and/or IP addresses or networks that are permitted or denied access to documents.

mod_auth_anon

This module mimics the behaviour of anonymous FTP - rather than having a database of valid credentials, it recognizes a list of valid usernames (i.e., the way an FTP server recognizes "ftp" and "anonymous") and grants access to any of those with virtually any password. This module is more useful for logging access to resources and keeping robots out than it is for actual access control.

mod_auth_dbm

Like **mod_auth_db**, save that credentials are stored in a DBM file.

mod_auth_db

This module is essentially the same as **mod_auth**, except that the authentication credentials are stored in a Berkeley DB file format. The directives contain the additional letters "DB" (e.g., AuthDBUserFile).

mod_auth_digest

Whereas the other discretionary control modules supplied with Apache all support Basic authentication, **mod_auth_digest** is currently the sole supporter of the Digest mechanism. It underwent some serious revamping in 1999. Like **mod_auth**, the credentials used by this module are stored in a text file. Digest database files are managed with the htdigest tool. Using **mod_digest** is much more involved than setting up Basic authentication; please see the module documentation for details.

Configuring authentication modules

The security modules are passed the information about which authentication databases to use via directives in the Apache configuration files, such as **AuthUserFile** or **AuthDBMGroupFile**. An alternate approach is the use of .htaccess files, which can be placed in the directories to be protected.

The first approach. The resource being protected is determined from the placement of the directives in the configuration files; in this example:

```
<Directory /home/johnson/public_html>
<Files foo.bar>
AuthName "Foo for Thought"
        AuthType Basic
        AuthUserFile /home/johnson/foo.htpasswd
```

```
        Require valid-user
</Files>
</Directory>
</screen>
```

The resource being protected is "any file named foo.bar" in the /home/johnson/public_html directory or anywhere underneath it. Likewise, the identification of which users are authorized to access foo.bar is stated by the directives -- in this case, any user with valid credentials in the /home/johnson/foo.htpasswd file can access it.

The other approach. Create an .htaccess file. The server behaviour can be configured on a directory-by-directory basis by using .htaccess files in directories accessed by the server (provided you configured this in the httpd.conf file using the AllowOverride option).

> **Note**
>
> By using an .htaccess file you can also restrict or grant access to certain user to documents in or under a directory. Documents that are to be restricted should be put in directories separate from those you want unrestricted.

The authentication part of an .htaccess file contains 2 sections:

The first section of .htaccess can contain lines to determine which authentication type to use. It can also contain the names of the password file to use, or the *group file* to use, e.g.:

```
AuthUserFile {path to passwd file}
AuthGroupFile {path to group file}
AuthName {title for dialog box}
AuthType Basic
```

The second section of .htaccess contains a section that defines the access rights for the current directory to ensure that only user {username} can access the current directory:

```
<Limit GET>
 require user {username}
</Limit>
```

The Limit section can contain other directives, that allow access from only certain IP addresses or only for users who are part of a certain set of users, a *group*.

As an example of the usage of both access control types (DAC and MAC), the following

would permit any client on the local network (IP addresses 10.*.*.*) to access the foo.html page without hindrance, but require a username and password for anyone else:

```
<Files foo.html>
Order Deny,Allow
        Deny from All
        Allow from 10.0.0.0/255.0.0.0
        AuthName "Insiders Only"
        AuthType Basic
        AuthUserFile /usr/local/web/apache/.htpasswd-foo
        Require valid-user
        Satisfy Any
</Files>
```

User files

The **mod_auth** module uses plain text user files. Its entries are of the form "username: password"; additional fields may follow the password, separated from it by a colon, but they're ignored. The password field should be encrypted. To create and update the flat-files used to store usernames and password for basic authentication of HTTP users, you can use the command **htpasswd**. This program can only be used when the usernames are stored in a flat-file. **htpasswd** encrypts passwords using either a version of MD5 modified for Apache, or the system's crypt() routine. It is permissible to have some user records using MD5-encrypted passwords, while others in the same file may have passwords encrypted with crypt (). To use a DBM database (as used by **mod_auth_db**) you may use **dbmmanage**. For other types of user files/databases, please consult the documentation that comes with the chosen module.

Group files

It is possible to deny or allow access for a group of users. This is done by creating a group file. It contains a list of groups and members. The group names are completely arbitrary. The usernames should occur in the password file. The format looks like this:

```
{group1}:{username1} {username2} etc...
{group2}:{username1} {username3} etc...
```

In this example, group1 and group2 are different group names and username1, username2, and username3 are usernames from the password file. Be sure to put each group entry on its own line.

Note

A username can be in more than one group entry. This simply means that the user is a member of both groups.

The last step is to make sure that the read permissions for the group file have been set for everyone (i.e., owner, group and other). Referral to this file is done by insertion of AuthGroupFile directives into either the master configuration file or into the .htaccess file.

Example 8.1. Example

To ensure that only users of the group mygroup, as defined in file /etc/httpd/groups/groupfile, can access a directory, you'd use these directives in .htaccess:

```
AuthGroupFile /etc/httpd/groups/groupfile
<Limit GET>
  require group mygroup
</Limit>
```

Configuring mod_perl

mod_perl is another module for Apache. You can configure it either at compile-time, or as a DSO. With **mod_perl** it is possible to write Apache modules entirely in Perl, letting you easily do things that are more difficult or impossible in CGI programs. In addition, the persistent Perl interpreter embedded in the module saves the overhead of starting an external interpreter. Another important feature is code-caching: modules and scripts are loaded and compiled only once, and for the rest of the server's life they are served from the cache. Thus, the server spends its time running only already loaded and compiled code, which is very fast.

You have full access to the inner workings of the web server and can intervene at any stage of request-processing. This allows for customized processing of (to name just a few of the phases) URI->filename translation, authentication, response generation and logging. There is very little run-time overhead.

The standard Common Gateway Interface (CGI) within Apache can be replaced entirely with Perl code that handles the response generation phase of request processing. **mod_perl** includes two general purpose modules for this purpose: Apache::Registry, which can transparently run existing perl CGI scripts and Apache::PerlRun, which does a similar job, but allows you to run "dirtier" (to some extent) scripts.

You can configure your **httpd** server and handlers in Perl (using PerlSetVar, and <Perl> sections). You can also define your own configuration directives.

There are many ways to install **mod_perl**, e.g. as a DSO, either using APXS or not, from

source or from RPM's. Most of the possible scenarios can be found in the Mod_perl GuidePerlRef01. As an example we describe a scenario for building Apache and **mod_perl** from source code.

You should have the Apache source code, the source code for **mod_perl** and have unpacked these in the same directory [13]. You'll need a recent version of **perl** installed on your system. To build the module, in most cases, these commands will suffice:

```
$ cd ${the-name-of-the-directory-with-the-sources-for-the-module}
$ perl Makefile.PL APACHE_SRC=../apache_x.x.x/src \
            DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
$ make && make test && make install
```

After building the module, you should also build the Apache server. This can be done using the commands:

```
$ cd ${the-name-of-the-directory-with-the-sources-for-Apache}
$ make install
```

All that's left then is to add a few configuration lines to httpd.conf (the Apache configuration file) and start the server. Which lines you should add depends on the specific type of installation, but usually a few LoadModule and AddModule lines suffice.

As an example, these are the lines you would need to add to use **mod_perl** as a DSO:

```
LoadModule perl_module modules/libperl.so
AddModule mod_perl.c
PerlModule Apache::Registry

Alias /perl/ /home/httpd/perl/
<Location /perl>
  SetHandler perl-script
  PerlHandler Apache::Registry
  Options +ExecCGI
  PerlSendHeader On
</Location>
```

The first two lines will add the **mod_perl** module when Apache boots. On boot, the PerlModule directive ensures that the named Perl module is read in too. This usually is a Perl package file ending in .pm. The Alias keyword reroutes requests for URIs in the form http://www.host.com/perl/file.pl to the directory /home/httpd/perl. Next, we define settings for that

location. By setting the SetHandler, all requests for a Perl file in the directory /home/httpd/perl now will be redirected to the perl-script handler, which is part of the Apache::Registry module. The next line simply allows execution of CGI scripts in the specified location. Any URI of the form http://www.host.com/perl/file.pl now will be compiled once and cached in memory. The memory image will be refreshed by recompiling the Perl routine whenever its source is updated on disk.

Configuring mod_php support

PHP is a server-side, cross-platform, HTML embedded scripting language. PHP started as a quick Perl hack written by Rasmus Lerdorf in late 1994. Over the next two to three years, it evolved into what we today know as PHP/FI 2.0. PHP/FI started to get a lot of users, but things didn't start flying until Zeev Suraski and Andi Gutmans suddenly came along with a new parser in the summer of 1997, leading to PHP 3.0. PHP 3.0 defined the syntax and semantics used in both versions 3 and 4.

PHP can be called from the CGI interface, but most common approach is to configure PHP in the Apache web server as a (dynamic DSO) module. To do this, you can either use pre-build modules extracted from RPM's or roll your own from the source code[14]. You need to **configure** the make-process first. To tell **configure** to build the module as a DSO, you need to tell it to use APXS:

```
./configure -with-apxs
```

.. or, in case you want to specify the location for the **apxs** binary:

```
./configure -with-apxs={path-to-apxs}/apxs
```

Next, you can compile PHP by running the **make** command. Once all the source files are successfully compiled, install PHP by using the **make install** command.

Before Apache can use PHP, it has to know about the PHP module and when to use it. The **apxs** program took care of telling Apache about the PHP module, so all that is left to do is tell Apache about .php files. File types are controlled in the httpd.conf file, and it usually includes lines about PHP that are commented out. You want to search for these lines and uncomment them:

```
Addtype application/x-httpd-php .php
```

.. and restart Apache by issuing the **apachectl restart** command.

To test whether it actually works, create the following page:

```
<HTML>
<HEAD><TITLE>PHP Test </TITLE></HEAD>
<BODY>
<?phpinfo( ) ?>
</BODY>
</HTML>
```

Notice that PHP commands are contained by <? and ?> tags. Save the file as test.php in Apache's htdocs directory and aim your browser at http://localhost/test.php. A page should appear with the PHP logo and additional information about your PHP configuration.

Configuring Apache server options

The httpd.conf file contains a number of sections that allow you to configure the behavior of the Apache server. A number of keywords/sections are listed below.

MaxKeepAliveRequests

: The maximum number of requests to allow during a persistent connection. Set to 0 to allow an unlimited amount.

StartServers

: The number of servers to start initially.

MinSpareServers, MaxSpareServers

: Used for server-pool size regulation. Rather than making you guess how many server processes you need, Apache dynamically adapts to the load it sees. That is, it tries to maintain enough server processes to handle the current load, plus a few spare servers to handle transient load spikes (e.g., multiple simultaneous requests from a single Netscape browser). It does this by periodically checking how many servers are waiting for a request. If there are fewer than MinSpareServers, it creates a new spare. If there are more than MaxSpareServers, some of the spares die off.

MaxClients

: Limit on total number of servers running, i.e., limit on the number of clients that can simultaneously connect. If this limit is ever reached, clients will be *locked out*, so it should *not be set too low*. It is intended mainly as a brake to keep a runaway server from taking the system with it as it spirals down.

---

[13] The **mod_perl** module can be obtained at perl.apache.org, the source code for Apache at www.apache.org

[14] The source code for PHP4 can be obtained at www.php.net

Prev | Next
Securing a DNS Server (2.207.3) | Home | Maintaining a Web Server (2.208.2)

Maintaining a Web Server (2.208.2)

Candidates should be able to configure Apache to use virtual hosting for web-sites without dedicated IP addresses. This objective also includes creating an SSL certification for Apache and creating SSL definitions in configuration files using OpenSSL. Also included is the customization of file access by implementing redirect statements in Apache's configuration files.

Key files, terms and utilities include:

httpd.conf

Resources: LinuxRef08; Engelschall00; Krause01; the **man** pages for the various commands.

Apache Virtual Hosting

Virtual Hosting is a technique that provides the capability to host more than one domain on one *physical* host. There are two methods to implement virtual hosting:

Name-based virtual hosting. With name-based virtual hosting, the server relies on the client (e.g. the browser) to report the hostname as part of the HTTP headers. Using this technique, many different hosts can share the same IP address.

IP-based virtual hosting. Using this method, each (web)domain has it's own unique IP address. Since one *physical* host can have more than one IP address, one host can serve more than one (web)domain. In other words: IP-based virtual hosts use the IP address of the connection to determine the correct virtual host to serve.

Name-based virtual hosting

Name-based virtual hosting is a fairly simple technique. You need to configure your DNS server to map each hostname to the correct IP address first, then configure the Apache HTTP Server to recognize the different hostnames.

> **Tip**
>
> Name-based virtual hosting eases the demand for scarce IP addresses. Therefore you should use name-based virtual hosting unless there is a specific reason to

choose IP-based virtual hosting, see IP-based Virtual Hosting.

To use name-based virtual hosting, you must designate the IP address (and possibly port) on the server that will be accepting requests for the hosts. This is configured using the NameVirtualHost directive. Normally any and all IP addresses on the server should be used, so, you can use * as the argument to NameVirtualHost. Note that mentioning an IP address in a NameVirtualHost directive does not automatically make the server listen to that IP address: there are two additional directives used to restrict or specify which addresses and ports Apache listens to:

- BindAddress is used to restrict the server to listening to a single address, and can be used to permit *multiple Apache servers on the same machine* to listen to different IP addresses;
- Listen can be used to make a *single Apache server* listen to more than one address and/or port.

The next step is to create a <VirtualHost> block for each different host you would like to serve. The argument to the <VirtualHost> directive should be the same as the argument to the NameVirtualHost directive (i.e., an IP address or * for all addresses). Inside each <VirtualHost> block, you will need, at minimum, a ServerName directive to designate which host is served and a DocumentRoot directive to show where in the filesystem the content for that host lives.

Suppose that both www.domain.tld and www.otherdomain.tld point to the IP address *111.22.33.44*. You then simply add the following to httpd.conf:

```
NameVirtualHost 111.22.33.44

<VirtualHost 111.22.33.44>
  ServerName www.domain.tld
  DocumentRoot /www/domain
</VirtualHost>

<VirtualHost 111.22.33.44>
  ServerName www.otherdomain.tld
  DocumentRoot /www/otherdomain
</VirtualHost>
```

In the simplest case, the IP address *111.22.44.33* can be replaced by * to match all IP addresses for your server. Many servers want to be accessible by more than one name. This is possible with the ServerAlias directive placed inside the <VirtualHost> section, if, for example, you add the following to the first <VirtualHost> block above

ServerAlias domain.tld *.domain.tld

then requests for all hosts in the domain.tld domain will be served by the www.domain.tld virtual host. The wildcard characters * and ? can be used to match names.

Tip

Of course, you can't just make up names and place them in ServerName or ServerAlias. You must first have your DNS server properly configured to map those names to the IP address in the NameVirtualHost directive.

Finally, you can fine-tune the configuration of the virtual hosts by placing other directives inside the <VirtualHost> containers. Most directives can be placed in these containers and will then change the configuration only of the relevant virtual host. Configuration directives set in the main server context (outside any <VirtualHost> container) will be used only if they are not overridden by the virtual host settings.

Now when a request arrives, the server will first check if it is requesting an IP address that matches the NameVirtualHost. If it is, then it will look at each <VirtualHost> section with a matching IP address and try to find one where the ServerName or ServerAlias matches the requested hostname. If it finds one, it then uses the configuration for that server. If no matching virtual host is found, then the first listed virtual host that matches the IP address will be used.

As a consequence, the first listed virtual host is the default virtual host. The DocumentRoot from the main server will never be used when an IP address matches the NameVirtualHost directive. If you would like to have a special configuration for requests that do not match any particular virtual host, simply put that configuration in a <VirtualHost> container and list it first in the configuration file.

IP-based virtual hosting

Despite the advantages of name-based virtual hosting, there are some reasons why you might consider using IP-based virtual hosting instead:

- Name-based virtual hosting cannot be used with SSL secure servers because of the nature of the SSL protocol.
- Some ancient clients are not compatible with name-based virtual hosting. For name-based virtual hosting to work, the client must send the HTTP Host header. This is required by HTTP/1.1, and is implemented by all modern HTTP/1.0 browsers as an extension.
- Some operating systems and network equipment implement bandwidth management techniques that cannot differentiate between hosts unless they are on separate IP addresses.

As the term *IP-based* indicates, the server must have a different IP address for each IP-based virtual host. This can be achieved by equipping the machine with several physical network connections or by use of virtual interfaces, which are supported by most modern operating systems (see system documentation for details on IP aliasing and the **ifconfig** command).

There are two ways of configuring Apache to support multiple hosts.

- by running a separate **httpd** daemon for each hostname
- by running a single daemon which supports all the virtual hosts.

Use multiple daemons when:

- there are security issues, e.g., if you want to maintain strict separation between the web-pages for separate customers. In this case you would need one daemon per customer, each running with different User, Group, Listen and ServerRoot settings;
- you can afford the memory and file descriptor requirements of listening to every IP alias on the machine. It's only possible to Listen to the "wildcard" address, or to specific addresses. So, if you need to listen to a specific address, you will need to listen to all specific addresses.

Use a single daemon when:

- sharing of the **httpd** configuration between virtual hosts is acceptable;
- the machine serves a large number of requests, and so the performance loss in running separate daemons may be significant.

Setting up multiple daemons

Create a separate **httpd** installation for each virtual host. For each installation, use the Listen directive in the configuration file to select which IP address (or virtual host) that daemon services:

```
Listen 123.45.67.89:80
```

You can use the domain name if you want to, but it is recommended you use the IP address instead.

Setting up a single daemon

For this case, a single **httpd** will service requests for the main server and all the virtual hosts. The VirtualHost directive in the configuration file is used to set the values of ServerAdmin, ServerName, DocumentRoot, ErrorLog and TransferLog or CustomLog configuration

directives to different values for each virtual host.

```
<VirtualHost www.smallco.com>
   ServerAdmin webmaster@mail.smallco.com
   DocumentRoot /groups/smallco/www
   ServerName www.smallco.com
   ErrorLog /groups/smallco/logs/error_log
   TransferLog /groups/smallco/logs/access_log
</VirtualHost>

<VirtualHost www.baygroup.org>
   ServerAdmin webmaster@mail.baygroup.org
   DocumentRoot /groups/baygroup/www
   ServerName www.baygroup.org
   ErrorLog /groups/baygroup/logs/error_log
   TransferLog /groups/baygroup/logs/access_log
</VirtualHost>
```

Customizing file access

Redirect allows you to tell clients about documents which used to exist in your server's namespace, but do not anymore. This allows you to tell the clients where to look for the relocated document.

```
Redirect {old-URI} {new-URL}
```

How to create a SSL server Certificate

Whilst installing OpenSSL, the program **openssl** has been installed on your system. This command can be used to create the necessary files.

More specifically these are:

- the *RSA private key file* is a digital file that you can use to decrypt messages sent to you. It has a *public component* which you distribute (via your *Certificate file*) and allows people to encrypt messages to you;
- a *Certificate Signing Request (CSR)* is a digital file which contains your public key and your name. You send the CSR to a *Certifying Authority (CA)* to be converted into a real *Certificate*.
- A *Certificate* contains your RSA public key, your name, the name of the CA and is digitally signed by your CA. Browsers that know the CA can verify the signature on that Certificate, thereby obtaining your RSA public key. That enables them to send

messages which only you can decrypt.

To create a RSA private key for your Apache server (it will be Triple-DES encrypted and PEM formatted):

```
$ openssl genrsa -des3 -out server.key 1024
```

openssl will ask for a pass-phrase. Please backup this server.key file and remember the pass-phrase.

To create a Certificate Signing Request (CSR) with the server RSA private key (output will be PEM formatted):

```
$ openssl req -new -key server.key -out server.csr
```

Make sure you enter the FQDN ("Fully Qualified Domain Name") of the server when OpenSSL prompts you for the "CommonName", i.e. when you generate a CSR for a web-site which will later be accessed via https://www.foo.dom/, enter "www.foo.dom" here.

You now have to send this Certificate Signing Request (CSR) to a Certifying Authority (CA) for signing. The result is a real Certificate which can be used for Apache. Here you have two options: First you can let the CSR sign via a commercial CA like Verisign or Thawte. Then, you usually have to post the CSR into a web form, pay for the signing and await a signed Certificate that you can store in a server.crt file. Secondly, you could create your own CA and sign the certificate yourself.

The server.csr file is no longer needed. Now you have two files: server.key and server.crt. In your Apache's httpd.conf file you should refer to them using lines like these:

```
SSLCertificateFile    /path/to/this/server.crt
SSLCertificateKeyFile /path/to/this/server.key
```

Tip

How do you run Apache-SSL as a shareable (DSO) module? First, configure the shared module support in the source tree:

```
./configure --enable-shared=apache_ssl
```

then enable the module in your httpd.conf:

```
LoadModule apache_ssl_module modules/libssl.so
```


Prev | Up | Next

Chapter 8. Web Services (2.208) | Home | Implementing a Proxy Server (2.208.3)

Implementing a Proxy Server (2.208.3)

Candidates should be able to install and configure a proxy server using **squid**. This objective includes implementing access policies, setting up authentication and utilizing memory usage.

Key files, terms and utilities include:

squid.conf
http_access

Resources: Kiracofe01; Brockmeier01; Wessels01; Pearson00; the **man** pages for the various commands.

Web-caches

A *web-cache*, also known as a *http proxy*, is used to reduce bandwidth demands and often allows for finer-grained access control. Using a proxy, the client specifies the hostname and port number of a proxy in his web browsing software. The browser then makes requests to the proxy, and the proxy forwards them to the origin servers. A proxy will use locally cached versions of web-pages if they did not expire yet and will validate client-requests.

Additionally, there are *transparent proxies*. Usually this is the tandem of a regular proxy and a redirecting router. In these cases, a web request can be intercepted by the proxy, transparently. That is, as far as the client software knows, it is talking to the originating server itself, when it is really talking to the proxy.

squid

squid is a high-performance proxy caching server for web clients. **squid** supports more then just HTTP data objects: it supports FTP and **gopher** objects too. **squid** handles all requests in a single, non-blocking, I/O-driven process. **squid** keeps meta data and, especially, hot objects cached in RAM, caches DNS lookups, supports non-blocking DNS lookups and implements negative caching of failed requests. **squid** supports SSL, extensive access controls and full request logging. By using the lightweight Internet Cache Protocol, **squid** caches can be arranged in a hierarchy or mesh for additional bandwidth savings.

squid can be used for a number of things, including saving bandwidth, handling traffic spikes and caching sites that are occasionally unavailable. **squid** can also be used for load balancing. Essentially, the first time **squid** receives a request from a browser, it acts as an

intermediary and passes the request on to the server. **squid** then saves a copy of the object. If no other clients request the same object, you'll see no benefit. However, if multiple clients request the object before it expires from the cache, **squid** can speed up transactions and save bandwidth. If you've ever needed a document from a slow site, say one located in another country, hosted on a slow connection or both, you can see the benefit of having a document cached. The first request may be slower than molasses, but the next request for the same document will be much faster, and the originating server's load will be lightened.

squid consists of a main server program **squid**, a Domain Name System lookup program **dnsserver**, some optional programs for rewriting requests and performing authentication, and some management and client tools. When **squid** starts up, it spawns a configurable number of **dnsserver** processes, each of which can perform a single, blocking Domain Name System (DNS) lookup. This reduces the amount of time the cache waits for DNS lookups.

squid is normally obtained in source code format. On most systems a simple **make install** will suffice. After that, you will also have a set of configuration files. All configuration files are, by default, kept in the directory /usr/local/squid/etc. However, the location may vary, depending on the style and habits of your distribution. The Debian packages, for example, place the configuration files in /etc, which is the normal home for .conf files. Though there is more than one file in this directory, only one file is important to most administrators, the squid.conf file. There are over a 125 option tags in this file - but you should only need to change eight options to get **squid** up and running. The other options give you additional flexibility.

squid assumes that you wish to use the default value if there is no occurrence of a tag in the squid.conf file. Theoretically, you could even run **squid** with a zero length configuration file. You will need to change at least one part of the configuration file though: the default squid.conf denies access to all browsers. You will need to edit the Access Control Lists to allow your clients to use the **squid** proxy. The most basic way to perform access control is to use the http_access option (see below).

Sections in the squid.conf file

http_port

this option determines on which port(s) **squid** will listen for requests. By default this is port 3128. Another commonly used port is port 8080.

cache_dir

used to configure specific storage areas. If you use more than one disk for cached data, you may need more than one mount point (for example /usr/local/squid/cache1 for the first disk, /usr/local/squid/cache2 for the second). **squid** allows you to have more than one cache_dir option in your config file. This option can have four parameters:

```
cache_dir /usr/local/squid/cache/ 100 16 256
```

The first option determines in what directory the cache should be maintained. The next option is a size value. **squid** will store up to that amount of data in that directory. The value is in megabytes and defaults to 100 Megabyte. The next two options set the number of subdirectories (first and second tier) to create in this directory. **squid** makes lots of directories and stores a few files in each of them in an attempt to speed up disk access (finding the correct entry in a directory with one million files in it is not efficient: it's better to split the files up into lots of smaller sets of files).

http_access, acl

The basic syntax of the option is http_access allow|deny [!]aclname. If you want to provide access to an internal network, and deny access to anyone else, your options might look like this:

```
acl home src 10.0.0.0/255.0.0.0
http_access allow home
```

The first line sets up an Access Control List class called "home" of an internal network range of addresses. The second line allows access to that range of addresses. Assuming it's the final line in the access list, all other clients will be denied. See also the section on acl.

Tip

Note that **squid**'s default behavior is to do the *opposite of your last access line* if it can't find a matching entry. For example, if the last line is set to "allow" access for a certain set of network addresses, then **squid** will deny any client that doesn't match any of its rules. On the other hand, if the last line is set to "deny" access, then **squid** will allow access to any client that doesn't match its rules.

authenticate_program

This option is used to specify which program to start up as a authenticator. You can specify the name of the program and any parameters needed.

redirect_program, redirect_children

The redirect_program is used to specify which program to start up as a redirector. The option redirect_children is used to specify how many processes to start up to do redirection.

After you have made changes to your configuration, issue **squid -k reconfigure** so that **squid** will recognize the changes.

Redirectors

squid can be configured to pass every incoming URL through a *redirector process* that returns either a new URL or a blank line to indicate no change. A redirector is an external program, e.g. a script that you wrote yourself. Thus, a redirector program is *NOT* a standard part of the **squid** package. However, some examples are provided in the contrib/ directory of the source distribution. Since everyone has different needs, it is up to the individual administrators to write their own implementation.

A redirector allows the administrator to control the locations to which his users may go. It can be used in conjunction with transparent proxies to deny the users of your network access to certain sites, e.g. porn-sites and the like.

The redirector program must read URLs (one per line) on standard input, and write rewritten URLs or blank lines on standard output. Also, **squid** writes additional information after the URL which a redirector can use to make a decision. The input line consists of four fields:

URL ip-address/fqdn ident method

- The URL originally requested.
- The IP address and domain name (if already cached by **squid**) of the client making the request.
- The results of any IDENT / AUTH lookup done for this client, if enabled.
- The HTTP method used in the request, e.g. GET.

A parameter that is not known/specified is replaced by a dash. A sample redirector input line:

ftp://ftp.gnome.org/pub/GNOME/stable/releases/ gnome-1.0.53/README 192.168.12.34/- - GET

A sample response:

ftp://ftp.net.lboro.ac.uk/gnome/stable/releases/ gnome-1.0.53/README 192.168.12.34/- - GET

It is possible to send the client itself an HTTP redirect to the new URL, rather than have **squid** silently fetch the alternative URL. To do this, the redirector should begin its response with 301: or 302: depending on the type of redirect.

A simple very fast redirector called **squirm** is a good place to start, it uses the **regex** library

to allow pattern matching.

The following Perl script may also be used as a template for writing your own redirector:

```
#!/usr/local/bin/perl
$|=1;
while (<>) {
  s@http://fromhost.com@http://tohost.org@;
  print;
}
```

Authenticators

squid can make use of authentication[15]. Authentication can be done on various levels, e.g. network or user.

Browsers are capable to send the user's authentication credentials using a special "authorization request header". This works as follows: if **squid** gets a request, given there was an http_access rule list that points to a proxy_auth ACL, **squid** looks for an *authorization header*. If the header is present, **squid** decodes it and extracts a username and password. If the header is missing, **squid** returns an HTTP reply with status 407 (Proxy Authentication Required). The user agent (browser) receives the 407 reply and then prompts the user to enter a name and password. The name and password are encoded, and sent in the Authorization header for subsequent requests to the proxy.

Security Issue

The HTTP protocol has two authentication modes: basic and digest mode. In digest mode, the server encodes the password with a different key in a unidirectional function and the client decodes the function using the password, then returns the key. This proves that the client knows the password, without actually transmitting the password at any point. However, most proxies are not capable of using digest mode, and are using basic mode instead. In basic mode, the name and password are encoded using *base64*. (See section 11.1 of RFC 2616). However, base64 is a binary-to-text encoding algorithm, it does NOT encrypt the information it encodes. This means that the username and password are essentially cleartext between the browser and the proxy. Therefore, you should not use the same username and password that you would use for your account login. **squid** supports digest mode. However, versions older than 2.5 need to be patched. The patch was integrated in version 2.5, January 2001.

Authentication is actually performed outside of the main **squid** process. When **squid** starts, it spawns a number of authentication subprocesses. These processes read usernames and passwords on stdin and reply with OK or ERR on stdout. This technique allows you to use a

number of different authentication schemes, although currently you can only use one scheme at a time.

The **squid** source code comes with a few authentcation processes. These include:

- LDAP: Uses the Lightweight Directory Access Protocol
- NCSA: Uses an NCSA-style username and password file.
- MSNT: Uses a Windows NT authentication domain.
- PAM: Uses the Linux Pluggable Authentication Modules scheme.
- SMB: Uses a SMB server like Windows NT or Samba.
- getpwam: Uses the old-fashioned Unix password file.

In order to authenticate users, you need to compile and install one of the supplied authentication modules, download a different one or write your own. You tell **squid** which authentication program to use with the authenticate_program option in squid.conf. You specify the name of the program, plus any command line options if necessary. For example:

authenticate_program /usr/local/squid/bin/ncsa_auth /usr/local/squid/etc/passwd

The **squid** source distribution includes a number of external authentication modules in the **auth_modules** directory. Additional modules that you might be interested in are:

RADIUS

This is Marc van Selm's RADIUS authenticator for **squid**.

Non-Anon LDAP

Karel De Bruyne has modified Alan Spark's original LDAP authenticator to work for non-anonymous LDAP servers.

Group LDAP

Tobias Crawley has a patch that supports static and dynamic LDAP group lookups when doing LDAP authentication. It modifies the core squid code to add support for ldap groups in acl mechanism, and adds a group ldap authentication module in the auth_modules/GROUP_LDAP/ directory.

squidauth.pl

This authentication program by Thomas Börnert is written in **perl** and uses the **md5** and **MIME::Base64** Modules. It has some security features, which may be necessary for firewalls.

Access policies

Many squid.conf options require use of Access Control Lists (ACLs). Each ACL consists of a name, type and value (a string or filename). Access control lists (ACLs) are often regarded as the most difficult part of the **squid** cache configuration: the layout and concept is not immediately obvious to most people. Additionally, the use of external authenticators and the default ACL adds to the confusion.

ACLs can be seen as definitions of resources that may or may not gain access to certain functions in the web-cache. Allowing the use of the proxy server is one of these functions.

To regulate access to certain functions, you will have to define an ACL first, and then add a line to deny or allow access to a function of the cache, using that ACL as a reference. In most cases the feature to allow or deny will be http_access, which allows or denies a web browser's to access the web-cache. The same principles apply to the other options, such as icp_access.

To determine if a resource (e.g. a user) has access to the web-cache, **squid** works its way through the http_access list from top to bottom. It will match the rules, until one is found that matches the user and either denies or allows access. Thus, if you want to allow access to the proxy only to those users whose machines fall within a certain IP range you would use the following:

```
acl ourallowedhosts src 192.168.1.0/255.255.255.0
acl all src 0.0.0.0/0.0.0.0

http_access allow ourallowedhosts
http_access deny all
```

If a user from 192.168.1.2 connects using TCP and request a URL, **squid** will work it's way through the list of http_access lines. It works through this list from *top to bottom*, stopping after the *first* match to decide which one they are in. In this case, **squid** will match on the first http_access line. Since the policy that matched is allow, **squid** would proceed to allow the request.

The src option on the first line is one of the options you can use to decide which domain the requesting user is in. You can regulate access based on the source or destination IP address, domain or domain regular expression, hours, days, URL, port, protocol, method, username or type of browser. ACLs can also require user authentication, specify an SNMP read community string, or set a TCP connection limit.

For example, these lines would keep all internal IPs off the Web except during lunchtime:

```
acl allowed_hosts 192.168.1.0/255.255.255.0
acl lunchtime MTWHF 12:00-13:00
http_access allow allowed_hosts lunchtime
```

The MTWHF string denotes the proper days of the week, where M specifies Monday, T specifies Tuesday and so on: WHFAS (Wednesday-Sunday). For more options have a look at the default configuration file **squid** installs on your system.

Another example is the blocking of certain sites, based on their domain names:

```
acl adults dstdomain playboy.com sex.com
acl ourallowedhosts src 196.4.160.0/255.255.255.0
acl all src 0.0.0.0/0.0.0.0

http_access deny adults
http_access allow ourallowedhosts
http_access deny all
```

These lines prevent access to the web-cache (http_access) to users who request sites listed in the adults ACL. If another site is requested, the next line allows access if the user is in the range as specified by the ACL ourallowedhosts. If the user is not in that range, the third line will deny access to the web-cache.

To use an authenticator, you have to tell **squid** which program it should use to authenticate a user (using the authenticate_program option in the squid.conf file), than you'll need to set up an ACL of type proxy_auth, and need to add a line to regulate the access to the web-cache, using that ACL:

```
authenticate_program /sbin/my_auth -f /etc/my_auth.db
acl name proxy_auth REQUIRED
http_access allow name
```

The ACL points to the external authenticator /sbin/my_auth. If a user wants access to the webcache (the http_access function), you would expect that (as usual) the request is granted if the ACL name is matched. *HOWEVER.. this is not the case!*

> **Authenticator** allow **rules act as** deny **rules!**
>
> If the external authenticator allowed access the allow rule actually acts as if it were a deny rule! *Any following rules are consequently checked too* until another matching ACL is found. In other words: the rule http_access allow name should be read as http_access deny !name. The exclamation mark signifies a negation, thus the rule http_access deny !name means: "deny access to users *not* matching the 'name' rule".

squid always adds a default ACL!

squid automatically adds a final rule to the ACL section that *reverses* the preceding (last) rule: if the last rule was an allow rule, a deny all rule would be added, and vice versa: if the last rule was a deny rule, an allow all rule would be added automatically.

Both warnings imply that if the example above is implemented as it stands, the final line http_access allow name implicitly adds a final rule http_access deny all. If the external authenticator grants access, *the access is not granted, but the next rule is checked* - and that next rule is the default deny rule if you do not specify one yourself! This means that properly authorized people would be *denied* access. This exceptional behavior of **squid** is often misunderstood and puzzles many novice **squid** administrators. A common solution is to add an extra lines, like this:

```
http_access allow name
http_access allow all
```

Utilizing memory usage

squid uses lots of memory. For performance reasons this makes sense since it takes much, much longer to read something from disk than it does to read directly from memory. A small amount of metadata for each cached object is kept in memory, the so-called StoreEntry. For **squid** version 2 this is 56-bytes on "small" pointer architectures (Intel, Sparc, MIPS, etc) and 88-bytes on "large" pointer architectures (Alpha). In addition, there is a 16-byte cache key (MD5 checksum) associated with each StoreEntry. This means there are 72 or 104 bytes of metadata in memory for every object in your cache. A cache with 1,000,000 objects therefore requires 72 MB of memory for metadata only.

In practice, it requires much more than that. Other uses of memory by **squid** include:

- Disk buffers for reading and writing
- Network I/O buffers
- IP Cache contents
- FQDN Cache contents
- Netdb ICMP measurement database
- Per-request state information, including full request and reply headers
- Miscellaneous statistics collection.
- *Hot objects* which are kept entirely in memory.

You can use a number of parameters in squid.conf to determine **squid**'s memory utilization.

The cache_mem parameter specifies how much memory to use for caching *hot* (very popular) requests. **squid**'s actual memory usage depends strongly on your disk space (cache space)

and your incoming request load. Reducing cache_mem will *usually* also reduce **squid**'s process size, but not necessarily.

The maximum_object_size option in squid.conf specifies the maximum file size that will be cached. Objects larger than this size will NOT be saved on disk. The value is specified in kilobytes and the default is 4MB. If speed is more important than saving bandwidth, you should leave this low.

The minimum_object_size option: objects smaller than this size will NOT be saved on disk. The value is specified in kilobytes, and the default is 0 KB, which means there is no minimum (and everything will be saved to disk).

The cache_swap option tells **squid** how much disk space it may use. If you have a large disk cache, you may find that you do not have enough memory to run **squid** effectively. If it performs badly, consider increasing the amount of RAM or reducing the cache_swap.

---

[15] The information here is current for version 2.4.

Chapter 9. File and Service Sharing (2.209)

Prev Next

Chapter 9. File and Service Sharing (2.209)

Revision $Revision: 1.5 $ ($Date: 2004/03/03 15:03:36 $)

This objective has a weight of 8 points and contains the following objectives:

Objective 2.209.1; Configuring a Samba Server (5 points)

> The candidate should be able to set up a Samba server for various clients. This objective includes setting up a login script for Samba clients and setting up an nmbd WINS server. Also included is the changing of the workgroup in which a server participates, defining a shared directory in smb.conf, defining a shared printer in smb.conf, using nmblookup to test WINS server functionality and using the smbmount command to mount an SMB share on a Linux client.

Objective 2.209.2; Configuring an NFS Server (3 points)

> The candidate should be able to create an exports file and specify filesystems to be exported. This objective includes editing exports file entries to restrict access to certain hosts, subnets or netgroups. Also included is specifying mount options in the exports file, configuring user ID mapping, mounting an NFS filesystem on a client and using mount options to specify soft or hard and background retries, signal handling, locking and block size. The candidate should also be able to configure tcp wrappers to further secure NFS.

Configuring a Samba Server (2.209.1)

Author: Willem Schreuder

Revision: $Revision: 1.9 $ ($Date: 2004/05/11 11:51:30 $)

Resources: Sharpe01; the **man** pages for the various commands.

What is Samba?

Samba implements the Server Message Block (or SMB for short) protocol. This is the protocol used by Microsoft to implement file and printer sharing. By installing Samba on a Linux machine, machines running the Windows Operating System and other platforms for which an SMB client is available can connect to the Linux machine and thus use files and printers available by the Linux machine.

Samba is available for many platforms including Linux, AIX, HP-UX, SunOS, FreeBSD, OS/2,

AmigaOS. Consult *Samba, Opening Windows To A Wider World*, for further information on platforms supporting Samba and for downloading a binary or source distribution for your platform.

Installing the Samba components

Depending on your distribution, you can

- get the sources and compile them yourself
- install the package using **rpm** (Red Hat, SuSE etc.)
- install the package using **apt-get** or **aptitude** (Debian)

Samba can be run either from **inetd** or as daemons. When run via **inetd** you save some memory and you can use tcpwrappers for extra security. When run as daemons, the server is always ready and sessions are faster.

If you want to use encrypted passwords, you will need to have a separate /etc/samba/smbpasswd file because the layout differs from /etc/passwd. During the installation, you can choose to have /etc/samba/smbpasswd generated from your /etc/passwd file. If you choose not to do this, you must use **smbpasswd** to set individual passwords for users.

Samba consists of two daemons: **nmbd** and **smbd**.

nmbd, the NetBIOS Name Service Daemon, handles NetBIOS name lookups and WINS requests. If you've told Samba to function as a WINS Server, an extra copy of **nmbd** will be running. Additionally, if DNS is used to translate NetBIOS names, another extra copy of **nmbd** will be running.

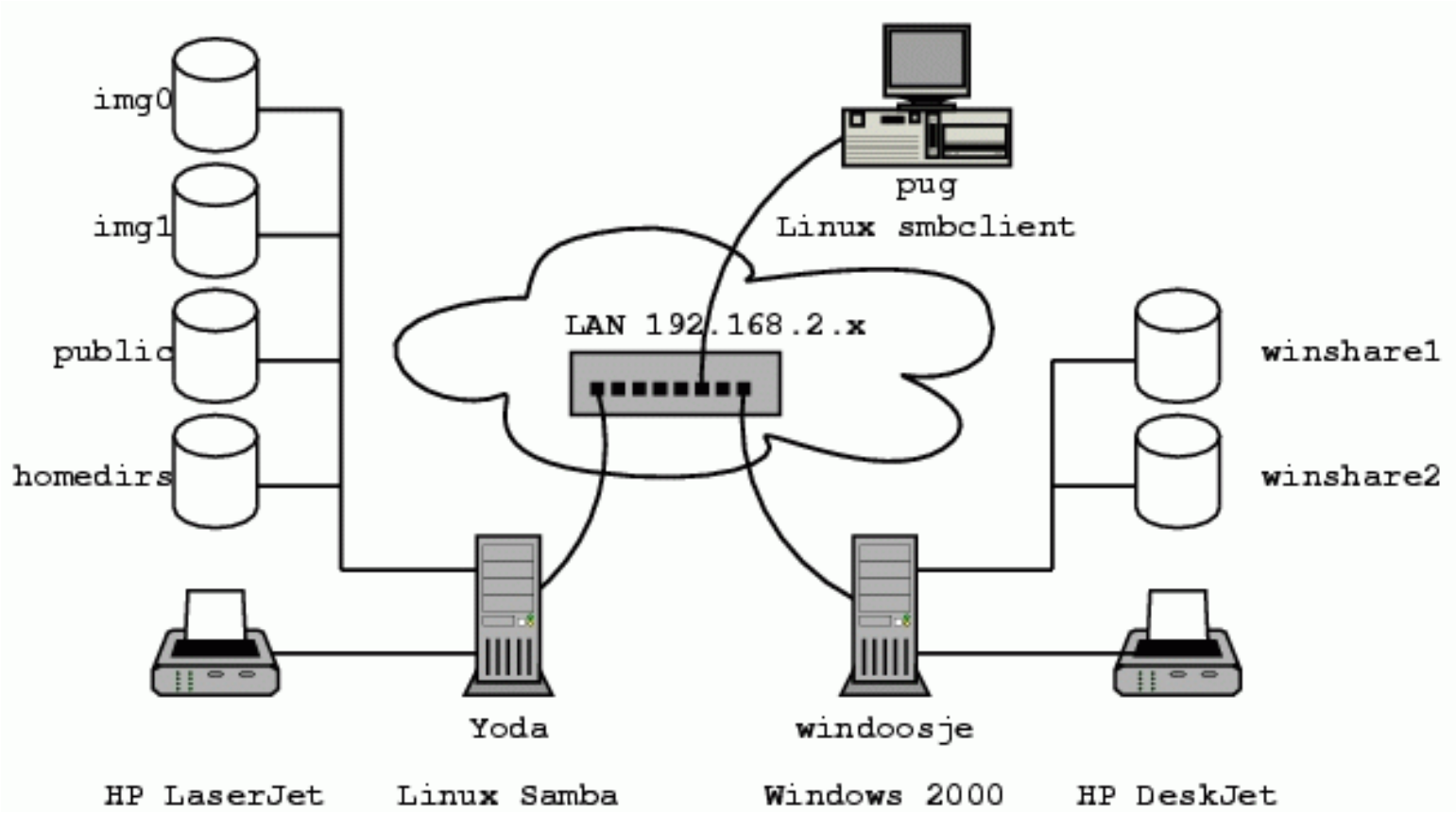
smbd, the Server Message Block Daemon, handles file and print access. For each client connected to the server, an extra copy of **smbd** runs.

Samba uses both the UDP and TCP protocols.

TCP is used for file and print sharing on port 139.

UDP is used for the registration and translation of NetBIOS names and for browsing the network. Port 137 is used for name service requests and responses. Port 138 is used for datagram services to transmit small amounts of data, such as server announcements.

An example of the functionality we wish to achieve



We've got three machines connected via a network. The machines "Yoda" and "windoosje" contain files the other machines must be able to manipulate. Also, the machines must be able to use each other's printers. "Yoda" is running Linux and has Samba installed. "windoosje" is running Microsoft Windows 2000 and "pug" is running Linux and has **smbclient** installed to be able to function as a Samba client.

We want to share just parts of the filesystems on "Yoda" and "windoosje".

The public share on "Yoda" should be available to everyone.

The img0 share on "Yoda" should be available to the user "willem" only.

The img1 share on "Yoda" should be available to both the users "willem" and "rc564".

The home directories on "Yoda" should be available to their respective users.

On the Windows 2000 machine we've got some generic files in the directories f:\winshare1 and f:\winshare2 we wish to make available to the Linux machine running Samba.

Accessing Samba shares from Windows 2000

Since I am using the Windows 2000 machine as a workstation, I didn't feel the need for domains, primary or otherwise. Instead, I have told the Windows 2000 machine that it is part of the workgroup "falcon".

I have included the /etc/samba/smb.conf file that contains the settings to make directories accessible from Windows 2000. Note that case doesn't matter to Microsoft Windows but does matter to Linux when writing workgroup names and machine names. Please read the comments before continuing:

```
#----------------------------------------------------------
# This is: /etc/samba/smb.conf
#
#----------------------------------------------------------

[global]
#----------------------------------------------------------
# This section contains the global server settings and the
# defaults that will be used for the parameters of the other
# sections if they are not specifically assigned other values
# in those other sections.
#
# Samba joins the FALCON workgroup
#----------------------------------------------------------
        workgroup = FALCON

# Describe the server to the clients
#----------------------------------------------------------
        server string = Linux Samba Server %L

# Only allow connections from machines on our LAN
#----------------------------------------------------------
        hosts allow = 192.168.2.0/255.255.255.0

# Windows 2000 uses encrypted passwords, so do we
#----------------------------------------------------------
        encrypt passwords = yes

# Tell Samba to use smbpasswd file for user validation
#----------------------------------------------------------
        security = user
        smb passwd file = /etc/samba/smbpasswd

# Make the server also available as Yoda1 to enable connection
# from Windows as another user
#----------------------------------------------------------
        netbios name = Yoda
        netbios aliases = Yoda1
```

```
# Access from clients will be logged in log.<NetBIOS name>
#----------------------------------------------------------
        log file = /var/log/samba/log.%m

[homes]
#----------------------------------------------------------
# This section enables the users that have an account and a
# homedirectory on the Linux Samba Server to access and modify
# the contents of that directory from a Samba client.
#
# Describe the share to the user
#----------------------------------------------------------
        comment = %U's homedirectory on %L from %m

# Do not show the homes share itself when browsing
#----------------------------------------------------------
        browsable = no

# Allow the user to write in his home directory
#----------------------------------------------------------
        writeable = yes

[public]
#----------------------------------------------------------
# This section defines a public share available for reading
# and writing for anyone on our LAN
#----------------------------------------------------------
        comment = Public Storage on %L
        path = /home/samba

# Show the public share when browsing
#----------------------------------------------------------
        browsable = yes

# Allow everyone to write in this directory
#----------------------------------------------------------
        writeable = yes

[img0]
#----------------------------------------------------------
# This section defines imaging share #0
#
# Describe the share to the user
#----------------------------------------------------------
        path = /img0
```

```
        comment = %U's Imaging Share #0 on %L from %m

# Show the img0 share itself when browsing
#----------------------------------------------------------
        browsable = yes

# Allow the user to write in his home directory
#----------------------------------------------------------
        writeable = yes

# Restrict access to valid users
#----------------------------------------------------------
        valid users = willem

[img1]
#----------------------------------------------------------
# This section defines imaging share #1
#
# Describe the share to the user
#----------------------------------------------------------
        path = /img1
        comment = %U's Imaging Share #1 on %L from %m

# Show the img1 share itself when browsing
#----------------------------------------------------------
        browsable = yes

# Allow the user to write in his home directory
#----------------------------------------------------------
        writeable = yes

# Restrict access to valid users
#----------------------------------------------------------
        valid users = willem,rc564
```

The sections [global], [homes] and [printers] are so called special sections.

The [global] section contains the parameters that are applicable for the whole server and the defaults that will be used for the parameters that are not mentioned in other sections.

The [homes] section makes it possible for users to connect to their home directories. The share name "homes" is changed by the server to the username. If you want to use some other directory instead of the user's home directory, you can do this by specifying the path. If you want to use the directory /home/sambahomes/<user> as the home directory, for instance, you can do this by setting the path parameter as follows:

```
path=/home/sambahomes/%S
```

The %S macro will be substituted by the user's name. Please consult the man page of smb.conf (**man smb.conf**) for information on the other macros that are available.

The [printers] section is used for giving access to the printers and will be described later in this chapter.

After creating the /etc/samba/smb.conf file, Samba must be restarted if it's already running or started if it's not:

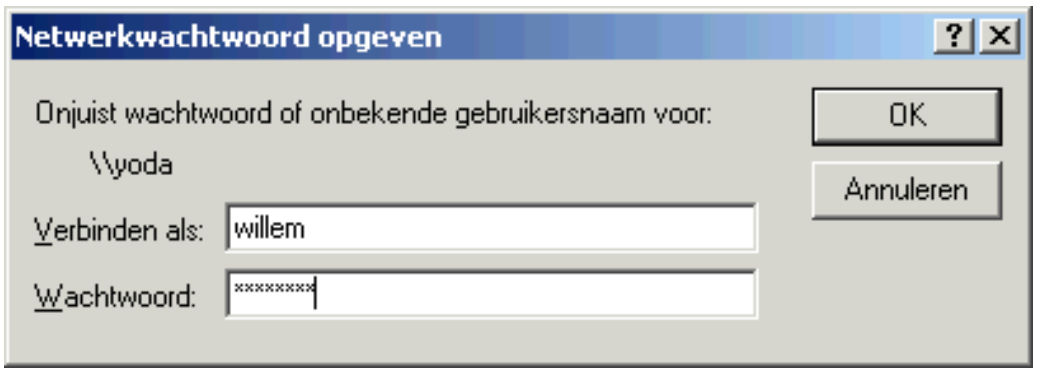
```
# /etc/init.d/samba restart or
# /etc/init.d/samba start
```

Now the samba passwords, which do not have to be identical to the Linux passwords, must be set. If a user already exists in the samba password file, you can use the command without the "-a" flag.

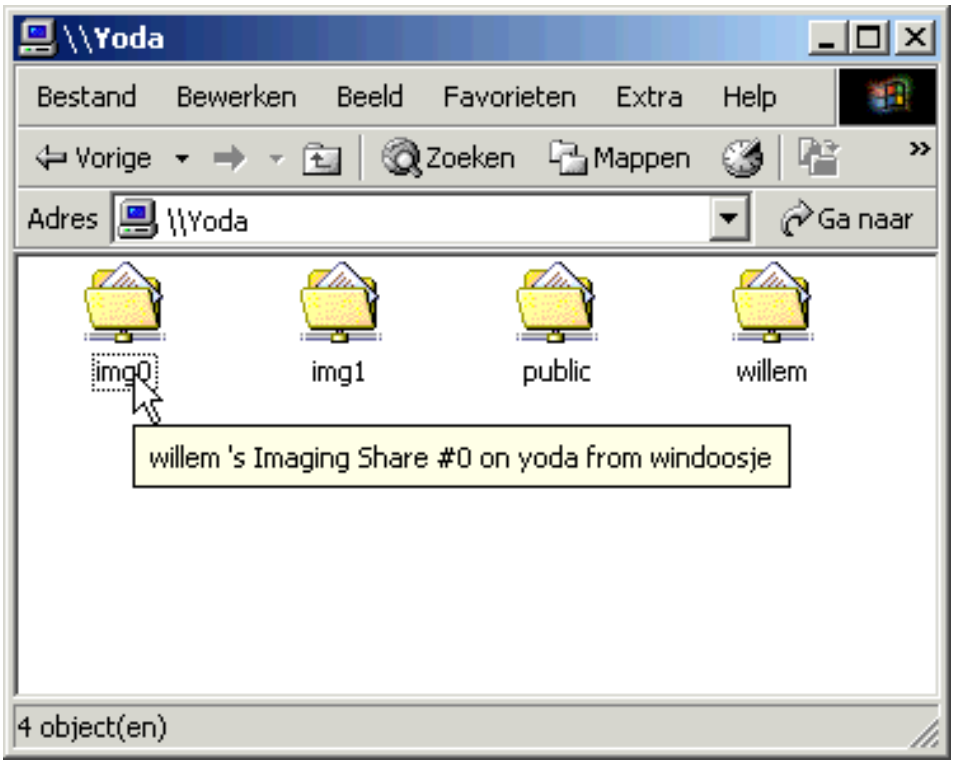
```
# smbpasswd [-a] user
```

Making the first connection from Windows 2000

Let's say you are the user "willem" on the Windows 2000 machine, and you enter "\\yoda" in the browser. You will be presented with a dialog - (illustrated below, but which you won't be able to read because it's in Dutch) - asking you to enter your username and accompanying password.



After entering the correct username and password, you will have access to your shares as shown below:



After opening the img0 share, another **smbd** process will be started by the already running **smbd** process:

```
# ps x -o pid,ppid,command

  PID  PPID COMMAND
    1     0 init [2]
...
26750     1 /usr/sbin/smbd -D
26753 26750 /usr/sbin/smbd -D
...
```

As you can see by looking at the process id (PID), the last **/usr/sbin/smbd** started is 26753 which has a process parent id (PPID) of 26750. This parent also is **/usr/sbin/smbd** and has a PPID of 1, which is the **init** process.

You can also use the **smbstatus** command to ask the system who is using which shares and which files are locked at the moment:

```
# smbstatus

Samba version 2.0.8
Service      uid      gid      pid      machine
----------------------------------------------
img0         willem   willem   26753    windoosje (192.168.2.11) Sat Feb 16
12:17:05 2002
```

No locked files

Share mode memory usage (bytes):
  1048464(99%) free + 56(0%) used + 56(0%) overhead = 1048576(100%) total

As you can see, the user "willem" is accessing the img0 share and has no files locked. You will probably almost never see file locks because their lifespan is so short. The file is only locked during saving. If you don't believe me, try this out with a file that takes several seconds to transport over the network, or "drag and drop" a complete directory, as I've done in the example that follows, to the img0 share while running the command **smbstatus -L**. The "-L" option will tell **smbstatus** to only show the locks:

while true; do smbstatus -L; sleep 1; done

No locked files

No locked files

No locked files

Locked files:
Pid DenyMode R/W Oplock Name

26753 DENY_ALL WRONLY EXCLUSIVE+BATCH /img0/Biljarten/2001-2002/
JoSterkRoosters.exe Sat Feb 16 13:12:51 2002

Locked files:
Pid DenyMode R/W Oplock Name

26753 DENY_ALL WRONLY EXCLUSIVE+BATCH /img0/Biljarten/2000-2001/
Arbiters.PX Sat Feb 16 13:12:52 2002

Locked files:
Pid DenyMode R/W Oplock Name

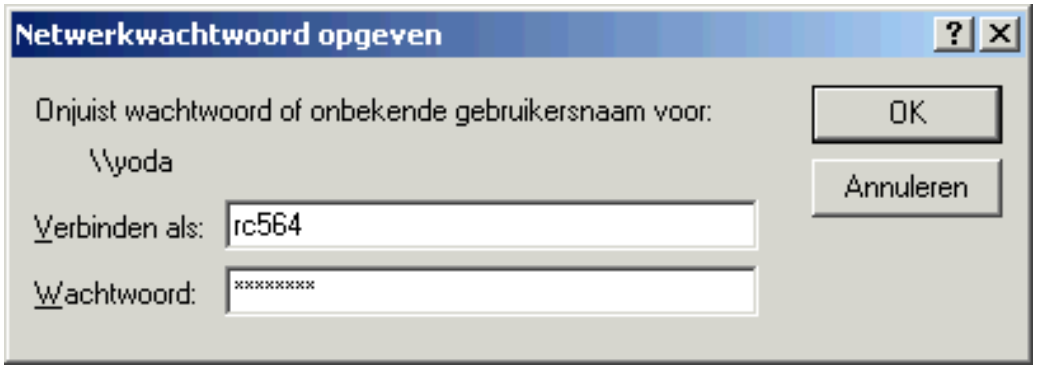
26753 DENY_ALL WRONLY EXCLUSIVE+BATCH /img0/Biljarten/2000-2001/
BasisForm112.~dfm Sat Feb 16 13:12:53 2002

...

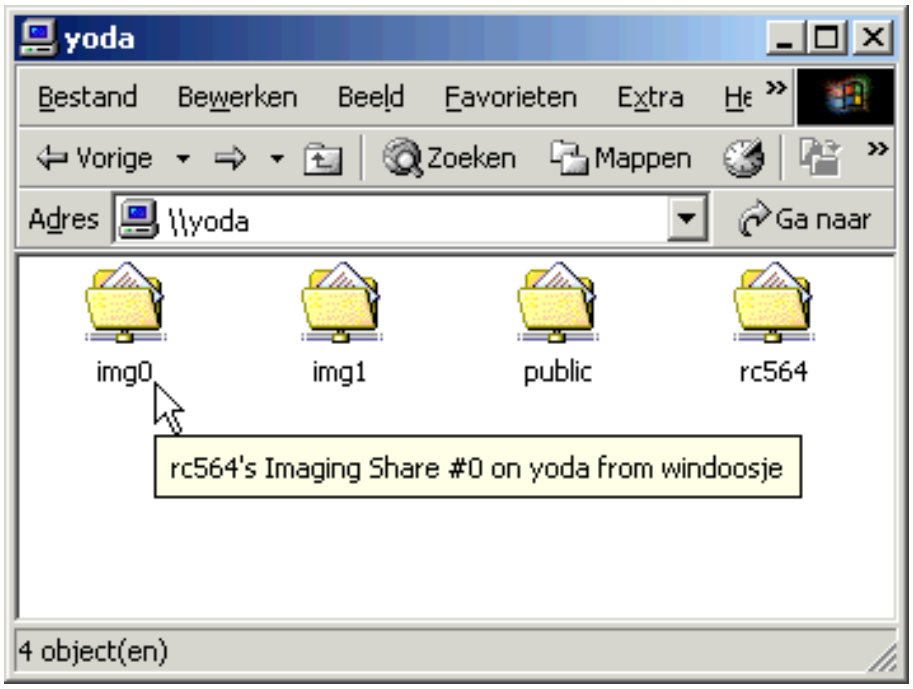
No locked files

Making the second connection from Windows 2000

Now let's see if the same works for the user rc564 by logging in to Windows 2000 as that user and entering "\\Yoda" in the browser:

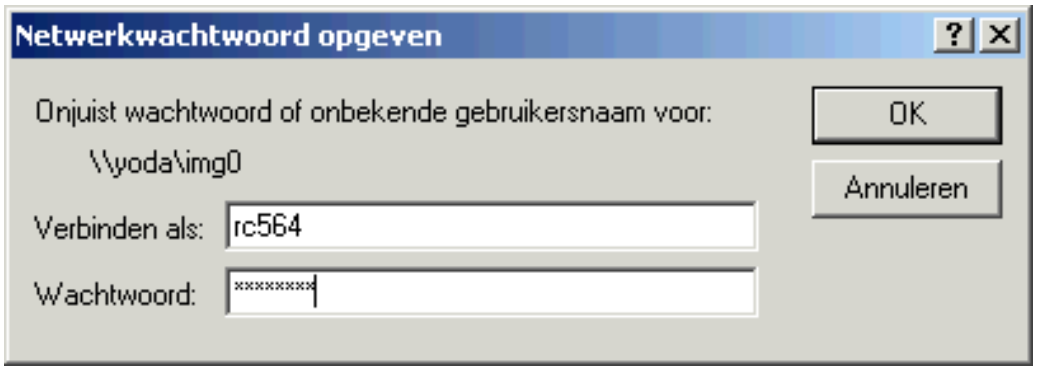


After entering the correct user and password combination, you will have access to your shares as shown below:

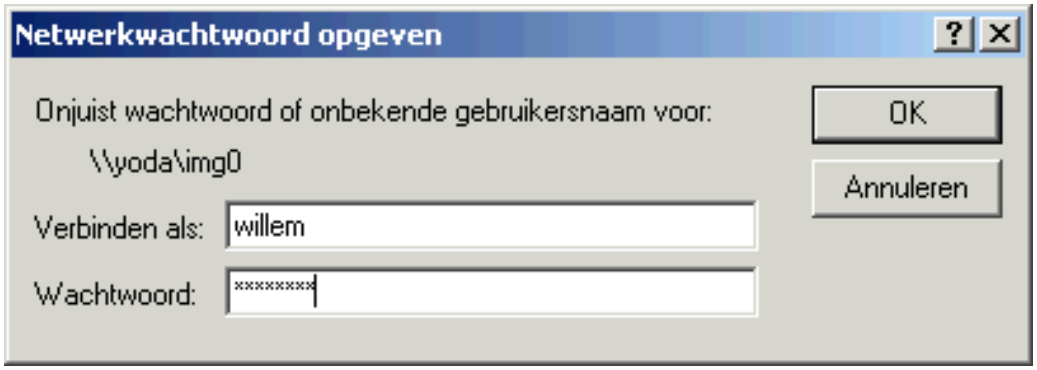


If everything is as it should be, the user "rc564" should not be able to write to the img0 share and should be able to write to the img1 share.

If you try to access the img0 share, a window will appear saying the password is wrong or that the username is unknown for the share. You will then have the opportunity to enter a username and password:



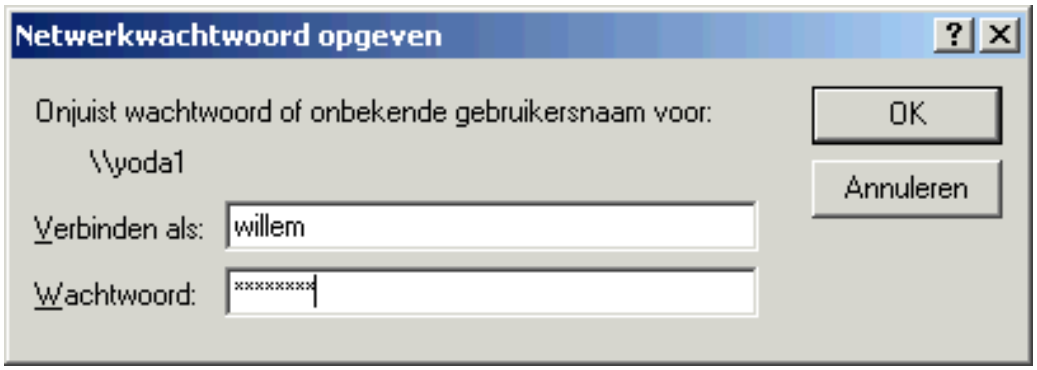
As expected, this doesn't work because the user "rc564" is not authorized to do this. But there is more to this than meets the eye. What if we were to connect as the user "willem" with the correct password? That should work, shouldn't it? Well, let's see:



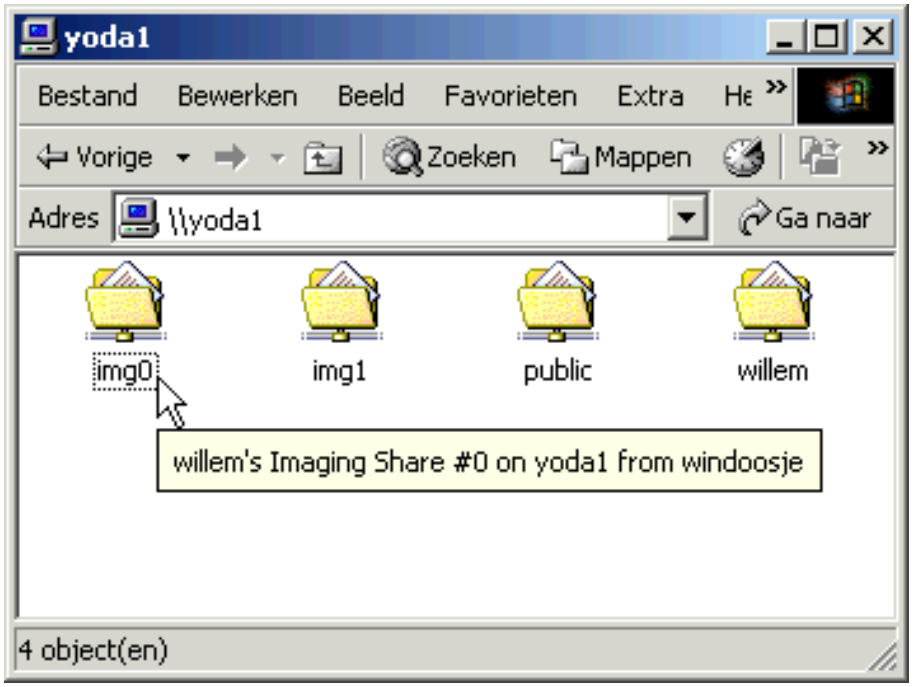
After hitting the "OK" button, we get the following response:

Which, translated, says that the share \\yoda\img0 is not accessible because the submitted set of references (username and password) is in conflict with an existing set of references.

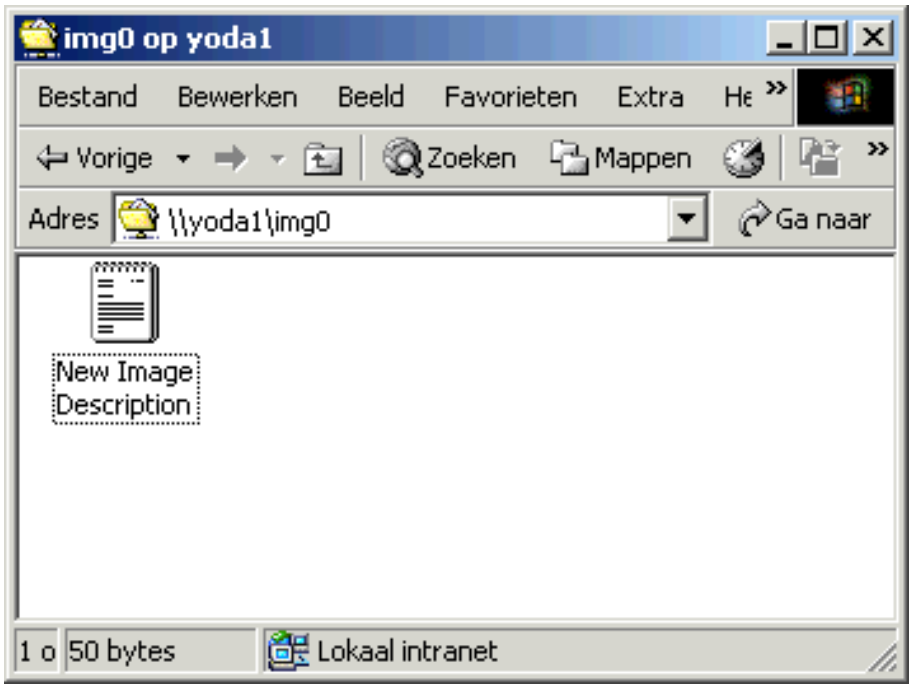
The cause of this seems to be that there already is a connection as "rc564" to Yoda. To prove it, let's connect to the server as the user "willem" by using the alias "\\Yoda1", which is a NetBios alias for "\\Yoda", while keeping the connection as the user "rc564" alive:



After hitting the "OK" button the next window appears showing that we've got a connection:



To prove that we also have write access, we create a text file:



Finally we use the command **smbstatus** to show that we really have two simultaneous connections:

```
Samba version 2.0.8
Service      uid      gid      pid      machine
----------------------------------------------
public       rc564    rc564    28305    windoosje (192.168.2.11) Sat Feb 16
13:48:35 2002
img0         willem   willem   28357    windoosje (192.168.2.11) Sat Feb 16
14:19:02 2002
```

Whether this is a Windows quirk or not will be demonstrated in the next section, where we'll try the same sequence from a Linux Samba client.

Accessing Windows or Samba shares from a Linux Samba client

With smbclient

The command **smbclient** implements an **ftp** like interface to the Samba shares.

You can use **smbclient** to find out which shares are available on the Windows machine (\\windoosje) by issuing the following command:

```
pug:~# smbclient -L windoosje -W falcon -U rc564
Password: ******
Domain=[FALCON] OS=[Windows 5.0] Server=[Windows 2000 LAN Manager]
```

```
    Sharename      Type      Comment
    ---------      ----      -------
    IPC$           IPC       Externe IPC
    F$           Disk      Standardshare
    ADMIN$         Disk      Remote Management
    C$           Disk      Standardshare
    WinShare#1     Disk      Word Documents
    WinShare#2     Disk      Excel Sheets
```

And on the Linux Samba Server \\Yoda as well:

```
pug:~# smbclient -L \\yoda -W falcon -U rc564
Password:  ******
Domain=[FALCON] OS=[Unix] Server=[Samba 2.0.8]

    Sharename      Type      Comment
    ---------      ----      -------
    public         Disk      Public Storage on yoda
    img0           Disk      rc564's Imaging Share #0 on yoda from pug
    img1           Disk      rc564's Imaging Share #1 on yoda from pug
    IPC$           IPC       IPC Service (Linux Samba Server yoda)
    rc564          Disk      rc564's homedirectory on yoda from pug

    Server             Comment
    ---------          -------
    YODA               Linux Samba Server yoda
    YODA1              Linux Samba Server yoda
```

Let's connect to \\windoosje\WinShare#1 to get a file:

```
pug:~# smbclient //windoosje/WinShare#1 -W falcon -U rc564
Password:  ******
Domain=[FALCON] OS=[Windows 5.0] Server=[Windows 2000 LAN Manager]
smb: \>
```

We've now got a connection. As with the **ftp** client, you can type "help" to find out which commands are available:

```
smb: \> help
?           altname       archive       blocksize     cancel
cd          chmod         chown         del           dir
du          exit          get           help          history
lcd         link          lowercase     ls            mask
```

```
md          mget        mkdir       more        mput
newer       open        print       printmode   prompt
put         pwd         q           queue       quit
rd          recurse     rename      rm          rmdir
setmode     symlink     tar         tarmode     translate
!
```

Finding out which files are available is done by issuing either the **ls** or the **dir** command:

```
smb: \> ls
  .                   D        0  Tue Feb 19 10:54:12 2002
  ..                  D        0  Tue Feb 19 10:54:12 2002
  Jo Sterk - Sponsoring.doc      A   107008  Sat Jul  7 11:05:34 2001
  Contributie_2001.doc           A    27648  Thu Jan 11 16:50:52 2001

                38461 blocks of size 262144. 37673 blocks available
smb: \>
```

As with an **ftp** client, you can download a file with the **get** or **mget** command:

```
smb: \> mget contr*
Get file Contributie_2001.doc? y
getting file Contributie_2001.doc of size 27648 as Contributie_2001.doc (52.9 kb/s) (average 52.9 kb/s)
smb: \>
```

As you can see, the command is case insensitive and wildcards can be used.

With smbmount

You can also mount the share as you would any other filesystem. This is done with the **smbmount** command. To be able to use the command **smbmount**, support for the SMB filesystem must be compiled into the kernel. You'll find the smbfs option in the filesystems section.

In the previous section, I promised to make connections from a Linux Samba client to the Linux Samba server as two different users to see if this can be done without using aliases. We'll try to make a connection as the user "willem" to his home directory on "\\Yoda" and as the user "rc564" to the public share. Here we go:

```
# mkdir /mnt/sh1        mountpoint for the first share
# mkdir /mnt/sh2        mountpoint for the second share
# smbmount //yoda/willem /mnt/sh1 -o username=willem
Password: *******
```

```
# smbmount //yoda/public /mnt/sh2 -o username=rc564
Password: ******
# mount
...
//yoda/willem on /mnt/sh1 type smbfs (0)
//yoda/public on /mnt/sh2 type smbfs (0)
```

So, this worked nicely. Let's ask the Samba Server which shares are in use at the moment:

```
# smbstatus

Samba version 2.0.8
Service      uid      gid      pid     machine
----------------------------------------------
willem     willem   willem   31345   pug     (192.168.2.8) Sun Feb 17
20:43:39 2002
public     rc564    rc564    31346   pug     (192.168.2.8) Sun Feb 17
20:44:30 2002

No locked files
```

As you can see, there are two connections being served by two separate processes. But, there is also a difference in how file-locking is handled. Remember that, when opening a file from Windows, there was no lock visible while the file was open - the lock was only present during the saving of the file. This is not the case when the Samba share is mounted by a Linux client. Opening the file hallo.txt in **vi** gives the following **smbstatus** result:

```
Samba version 2.0.8
Service      uid      gid      pid     machine
----------------------------------------------
willem     willem   willem   31345   pug     (192.168.2.8) Sun Feb 17
20:43:39 2002
public     rc564    rc564    31346   pug     (192.168.2.8) Sun Feb 17
20:44:30 2002

Locked files:
Pid    DenyMode   R/W       Oplock          Name
------------------------------------------------
31346  DENY_NONE  RDONLY    NONE          /home/samba/hallo.txt   Sun Feb
17 20:58:18 2002
```

As you can see, the file is immediately locked when opened.

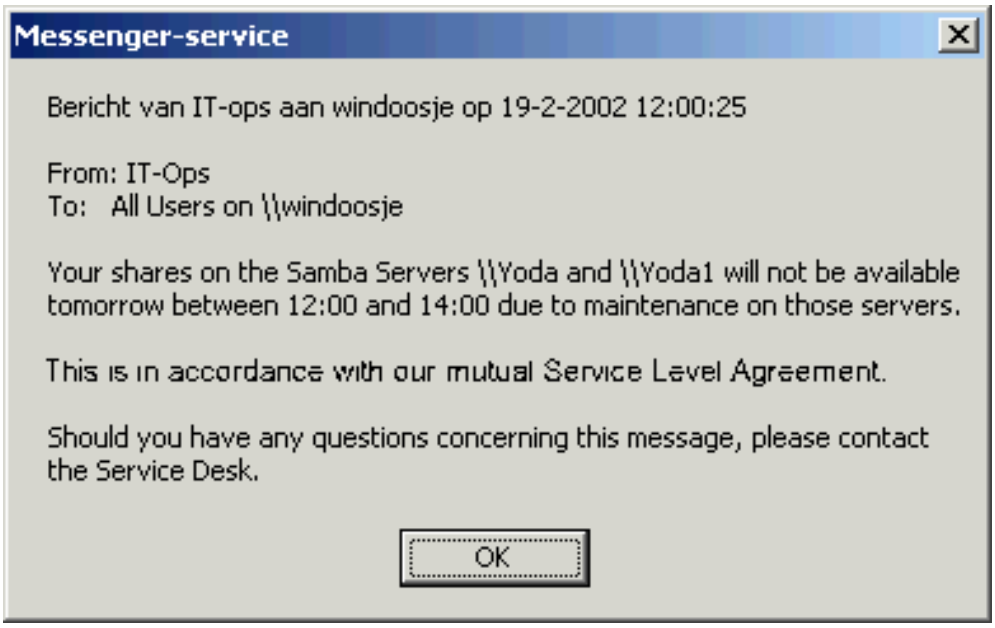
Sending a message with smbclient

Any machine running **WinPopup** can receive a message sent via the Winpopup protocol.

Assume, for a moment, that we've got to do maintenance on \\Yoda, and we wish to tell the user on \\windoosje that his shares will not be available during a certain period of time. Make a text file containing the message — I used **vi** to create a file called msg.txt — and use **smbclient** to send it as follows:

```
# cat msg.txt | smbclient -M windoosje -U IT-ops
```

The user on \\windoosje is presented with the following message:



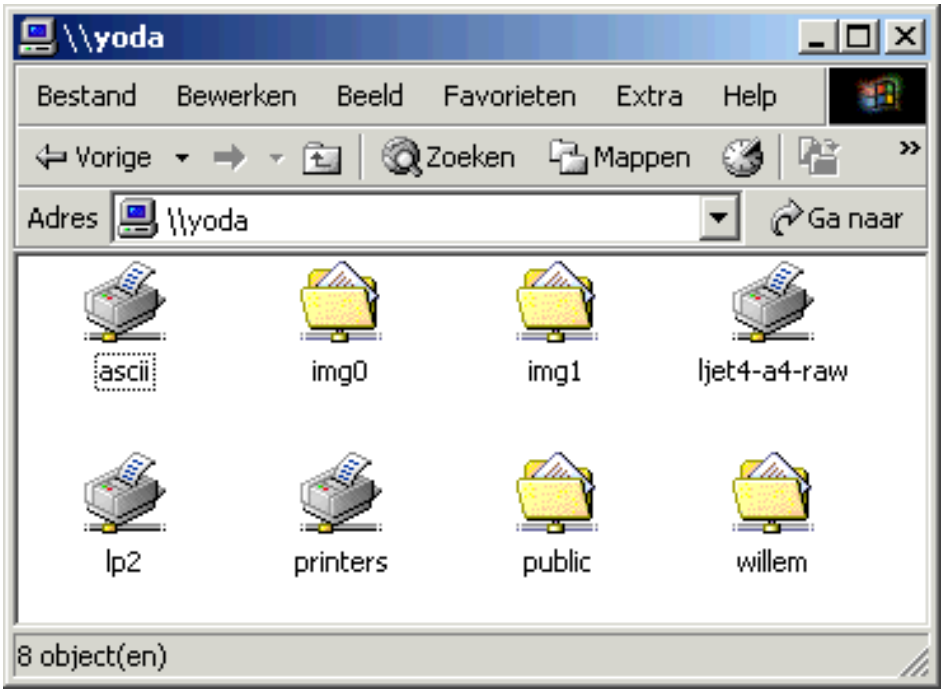
Using a Linux Samba printer from Windows 2000

Using Samba

To instruct Samba to share all printers defined in /etc/printcap, you may add a [printers] section to /etc/samba/smb.conf:

```
[printers]
  comment = Printer %p on Yoda
  path = /var/spool/samba
  printable = yes
```

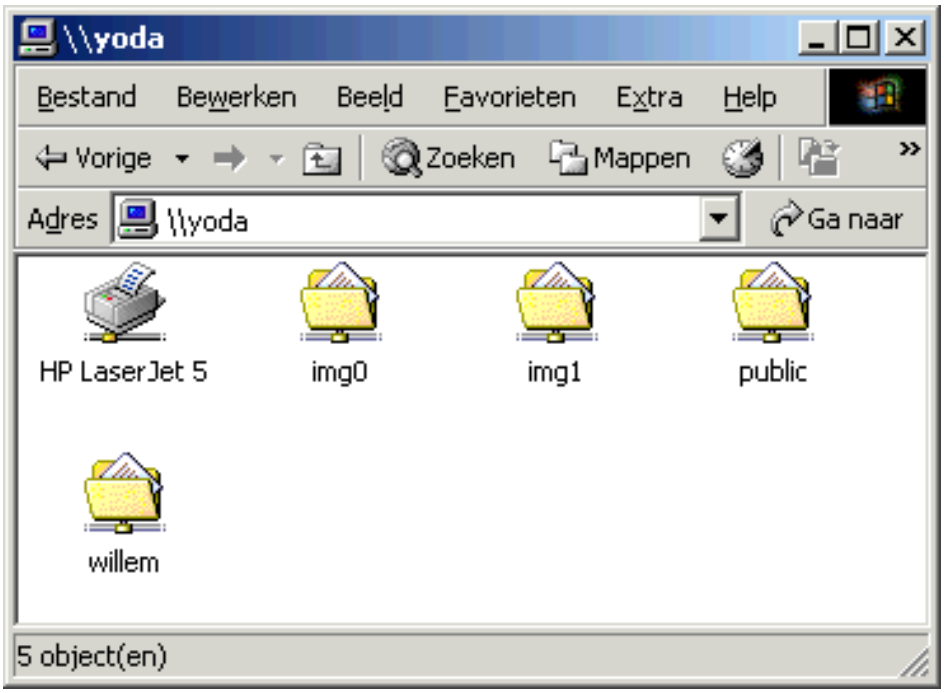
After restarting Samba by issuing the command **/etc/init.d/samba restart** we connect from Windows 2000 using the user "rc564" - which also exists on the Samba Server and must have the same password - to "\\Yoda" and get the following result:



Oops! That gave us four printers, and we've only got one. This happened because of the aliases in /etc/printcap. Our purpose was to get just one printer. This can be achieved by removing the [printers] section and replacing it with a printer-specific section:

```
[HP LaserJet 5]
  printer name = lp
  comment = HP LaserJet 5 on Yoda
  path = /var/spool/lpd/samba
  printable = yes
  writeable =  no
```

After restarting Samba we reconnect to \\Yoda and get the following result:

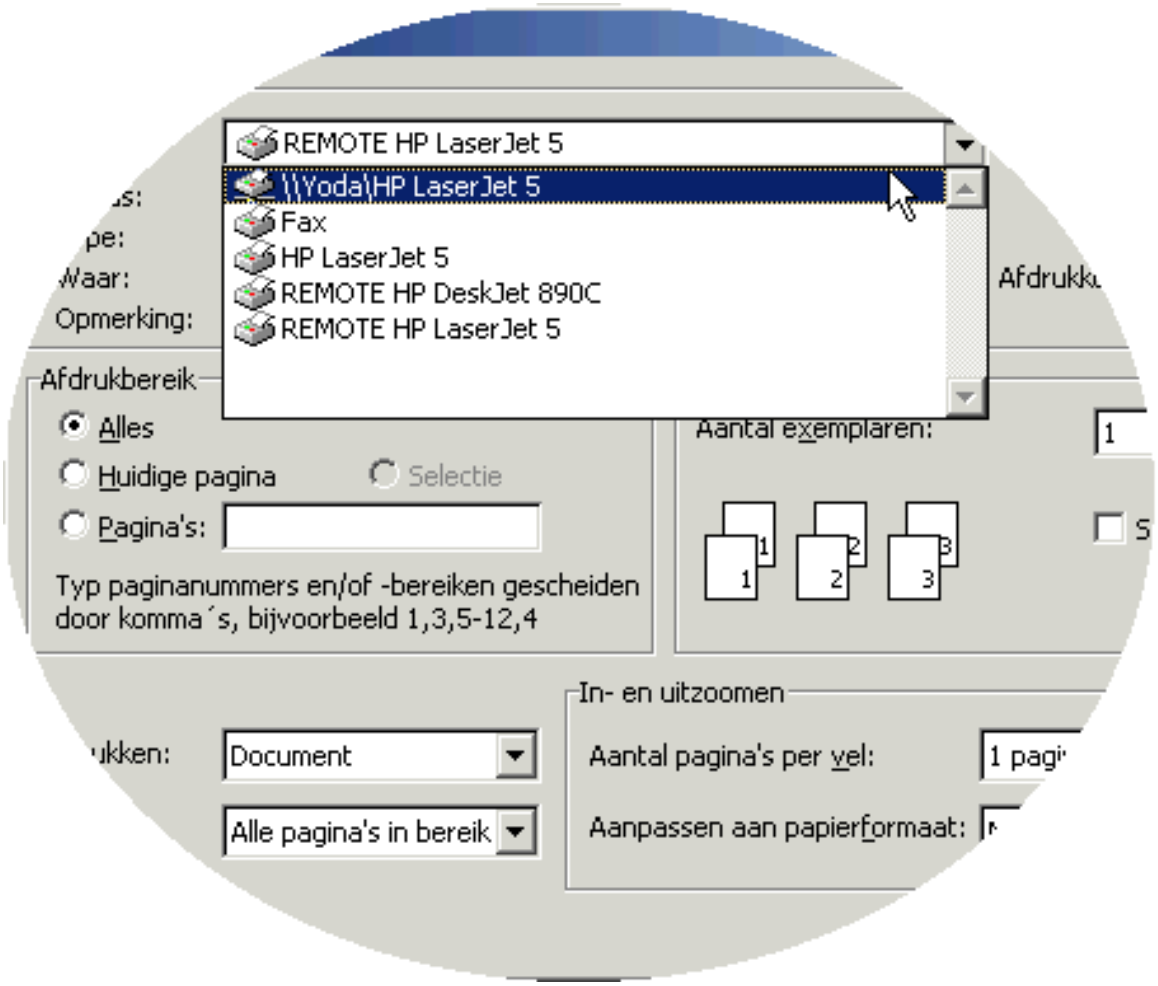


Now double-click on "HP LaserJet 5" and Windows will tell you that the printer has to be installed before you can use it and offers to go ahead with this. Allow Windows to install the printer.

Windows then says that there is a wrong printer driver installed on the machine to which the printer is connected and asks if the driver should be installed on the local computer. Allow Windows to do so.

Windows shows a dialog with manufacturers and printers, we choose HP and HP LaserJet 5, after which Windows installs the driver, and we are done.

Now let's see if it works. Open a document, for instance in MS Word, activate the print dialog to select the printer:



After hitting the "OK" button, the output will be sent to the printer.

If this doesn't work, chances are that the user has no write permissions in the /var/spool/lpd/samba directory. I experienced this problem myself and had the choice to either add all users able to print to the "lp" group or give all users write-access in the directory /var/spool/lpd/samba. I chose the latter, which is fine because the parameter setting "writeable = no" in the [printers] section of the Samba configuration file /etc/samba/smb.conf makes sure that no non-printing process can write in that directory.

What actually takes place is that the output is spooled to the directory set by the "path =" parameter which in this case is the directory /var/spool/lpd/samba. From there the output is spooled to the "real" spool directory of the printer as set by /etc/printcap. In this case, /var/spool/lpd. Check this out by "capturing" the spooled file as follows:

```
# while true; do ls -l samba >> t; ls -l lp >> t; echo "---" >> t ; done
The /var/spool/lpd/samba directory:
-rwxr--r--    1 rc564    lp          7480 Feb 19 17:35 RC564.aCKx75

The /var/spool/lpd/lp directory:
-rw-rw----    1 lp       lp            78 Feb 19 17:35 cfA002yoda
-rw-rw----    1 rc564    lp          7480 Feb 19 17:35 dfA002yoda
```

Issuing the **smbstatus** command shows the access to the share:

```
yoda:/var/spool/lpd# smbstatus

Samba version 2.0.8
Service      uid      gid      pid      machine
----------------------------------------------
HP LaserJe   rc564    rc564    31391    windoosje (192.168.2.11) Tue Feb 19 17:46:03 2002
```

Using lpr

Although this is not part of the exam, I feel that to be complete it is necessary to show you that a Linux printer can also be used from Windows 2000 without the need for Samba.

On the Windows machine the first thing to do is install an extra network component called "Print Services For Unix". This adds the ability to print to an **lpr** port.

The next thing to do is add a *local* printer — yes, you read it right: *not* a network printer.

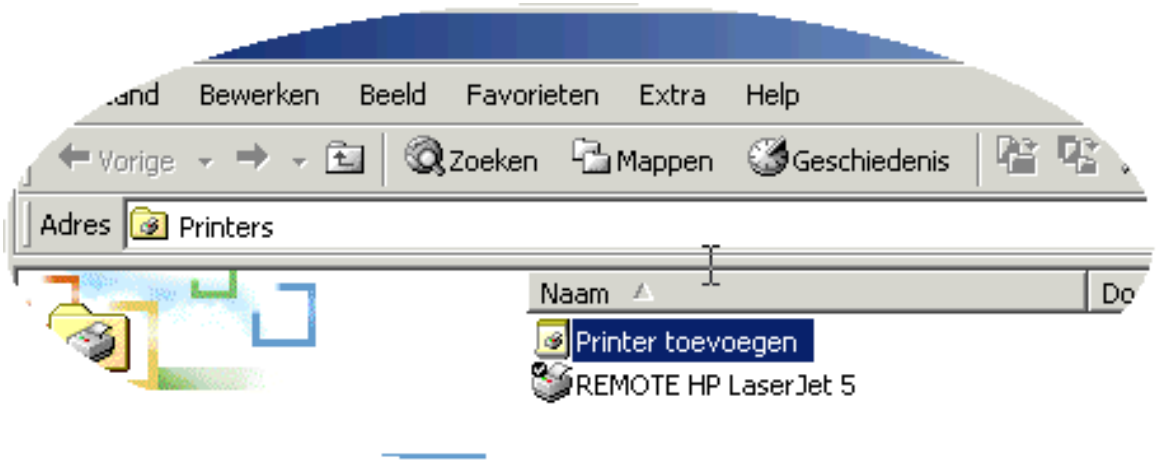
When Windows asks you to select the port, select the option that enables you to create a new port and select the type "LPR Port". You will then be presented with the next dialog which asks you to give the address of the **lpd** server and the name of the printer. Enter the Fully Qualified Domain Name of the printer.

Since I've called my local domain "falcon" and the machine to which my HP LaserJet 5 is connected is called "Yoda", this becomes *yoda.falcon*. The queue is called *lp* so that's what we enter for the printer name.

Then select the appropriate printer driver and give the printer a name. Let's call the printer "REMOTE HP LaserJet 5".

We are then presented with the possibility of sharing the printer under Windows. If you choose to do so and there are other Windows machines in the same workgroup such as Windows 98 for instance, they will see this printer as a Windows shared printer and can print to it. Windows 2000 will then send the output to the lp queue on Yoda. We don't need this functionality.

After printing a test page, we're done. Now have a look at the printers Windows 2000 knows:



The choice is yours: Samba printing or lpr printing. If printing is all you want to do, *and* you don't need the other functionality of Samba, *and* Print Services For Unix comes with your Windows OS, lpr printing may be the best choice for you.

Using a Windows printer from Linux

Although it is not in the topics I felt the need to describe this. Imagine a situation, for instance, where there are lots of Windows users and just a few Linux users. Wouldn't it be nice to be able to use the Windows printer from Linux without having to move the printer to a Samba server ?

Be careful with the sharename you define for the printer under Windows 2000. If Windows 2000 tells you that the printer might not be available to MS-DOS clients, shorten the name of the share until Windows 2000 does not complain anymore.

smbclient did *not* report the printer when it was called "HP DeskJet 890 C", but it did report the printer as soon as I called it "DeskJet_890C".

I have tried several utilities to accomplish this, amongst which are: **smbclient**, **smbprint**, **smbspool** and **apsfilter**.

And the winner is .... **apsfilter** for its ease of installation. The author, Andreas Klemm, only asks that you send him a postcard because he's interested in who is using **apsfilter**. I'll walk you through the installation:

```
# apsfilterconfig
    _/_/                _/_/ _/ _/   _/
  _/   _/ _/_/_/    _/_/_/  _/      _/ _/_/_/_/  _/_/  _/ _/_/
 _/_/_/_/ _/   _/ _/_/    _/_/_/_/ _/ _/   _/    _/_/_/_/ _/_/
_/   _/ _/   _/    _/_/  _/    _/ _/  _/    _/     _/
_/   _/ _/_/_/  _/_/_/    _/    _/ _/    _/_/  _/_/_/ _/
     _/
    _/
```

...

Accept license [Y|y|J|j|N|n] ?

Type "Y", hit "enter" and read the informational screens that follow.

Now we are checking file permissions in spooldir

Your line printer scheduler's spooldir seems to be: /var/spool/lpd

drwxrwsr-x 9 lp lp 4096 Feb 20 13:53 /var/spool/lpd

The Owner of your spooldir seems to be: lp
The Group of your spooldir seems to be: lp

Is this correct? [y/n]

Type "y" and hit "enter".

creating a working copy of printcap -> /etc/printcap.old

It seems you have configured a printer with this script before.

Do you want to (a)dd another printer entry or
to (o)verwrite the existing entries?
a/o?

If you've already defined other printers, as I have, type "a" and hit "enter".

```
==============================================================================
 A P S F I L T E R  S E T U P              -- MAIN MENUE --
==============================================================================

                                currently selected
------------------------------------------------------------
(D)   Available Device Drivers in your gs binary
(R)   Read Ghostscript driver documentation      (devices.txt)
(1)   Printer Driver Selection                   []
(2)   Interface Setup                        []

For printing the test page:
(3)   Paper Format (mandatory)               [a4]
(4)   Print Resolution in "dots per inch"    [default]
```

(5) Toggle Monochrome/Color (1bpp=b&w) [default]
(T) Print Test Page, on local or Windows remote prt.(after 1-5)
(V) View perf.log (times of print attempts)

(A) Abort installation (don't do anything)
(I) ==> Install printer with values shown above - repeat this step for installing multiple printers
(Q) ==> Finish installation

Your choice?

First, we have to select a driver for the HP DeskJet 890C. Type "1" and hit "enter".

```
==========================================================================
                PRINTER DRIVER SELECTION
==========================================================================
```

Please select the type of printer you want to install:

1) PostScript printer
2) printer driver natively supported by ghostscript

3) gimp-print / stp
4) hpdj
5) pcl3 (successor to hpdj, experimental)
6) IBM Omni
7) cdj880, cdj970
8) PPA printer, needs ghostscript "ppmraw" device and pnm2ppa

0) return to main menu

Your choice:

Choose "4" and hit "enter". You can then browse through a list of printer drivers. Remember the number of the correct driver (in my case, 131, comes closest to my printer). Hit "q" to close the list, type the number you remembered and hit "return" which takes us back to the main menu.

The next thing to do is to set up the interface — choose "2" and hit "return"

```
-------------------------------------------------------------
A P S F I L T E R  S E T U P          -- Interface Setup --
-------------------------------------------------------------
```

The easiest way to connect a printer to your computer is by

using the parallel interface, because it's usually *faster*, more standardized and therefore much easier to configure.

When configuring a serial printer, the installation dialogue asks you many questions about how to configure the serial interface of your computer, so that it works well with your printers current settings.

When using the serial interface, then you have to choose special cables, depending on the communication protocol between computer and printer (hardware/software handshaking). Many pitfalls here !

currently selected: Interface: [samba]
Device: [windoosje]
configure local / remote printer
1) local parallel/USB 2) local serial
3) Unix/network printer (lpd) 4) Windows / NT (samba)
5) AppleTalk

Your choice?

As you can see, there is a separate option for Samba. Choose "4" and hit "return". You will then be asked several questions as shown below:

A P S F I L T E R Samba Printer SETUP

Take care that smbclient is in apsfilters search path.
You can fine tune paths in /etc/apsfilter/apsfilterrc.
See smbclient manual page for more options if needed.

currently selected:
NetBIOS name of Windows Server :[ ]
Windows Server IP :[ ]
Printer Share Name :[ ]
Workgroup :[ ]
Windows Username :[ ]
Windows Password :[ ]

(you can fine tune some more values in the smbclient.conf file in the printers spool directory later)

NetBIOS name of Windows Server: windoosje

```
Windows Server IP Address    : 192.168.2.11
Printer Share Name           : DeskJet_890C
Workgroup Name               : falcon
Print as Windows GUEST user (no: use real account)? [y/n] n
Windows Username             : rc564
Windows Password             : thepassword
```

Now, the default papertype must be set. Choose "3", hit "return", and you'll be presented with a list from which you can choose:

```
    ------------------------------------------------------------
    A P S F I L T E R   S E T U P              -- Paper Format --
    ------------------------------------------------------------

    What paper format do you want to use for printing?

    1) DIN A4
    2) DIN A3
    3) US letter
    4) US legal
    5) US ledger

Your choice?
```

I chose "1", DIN A4. Now we are ready to print a test page. Choose "T", and hit "return", read the information and choose "T" again. You will then be asked if it is ok to print the testpage:

```
Printing Test page using: cat setup/test.ps | gs -q -sDEVICE=cdj890       \
-sPAPERSIZE=a4 -dNOPAUSE -dSAFER -sOutputFile='/tmp/aps_testout.iESShW'  -
Ok to print testpage? [y/n]
```

Type "y" and hit "return". The testpage will be created — which may take some time — and sent to the printer. If the output looks ok, choose "I", followed by "return", to install the printer with the values shown in the menu:

```
    ================================================================================
            Filter installation -- final steps
    ================================================================================

It's recommended to have one 'raw' entry for each physical printer.
If you're not sure, say 'y' -- it won't hurt.

Do you want me to create one for printer at windoosje? (y/n)
```

A Ok, let say "y" here.

Please enter a printer queue name for printer 'cdj890'.
The default name is 'auto3'.

Your choice:

Let's call the printer "dj890c".

** creating printcap entry for printer dj890c...
  creating spooldir ...
  creating samba config file ...
  read protect password information...
  remember SETUP settings in printers apsfilterrc file...

Please enter a printer queue name for printer 'cdj890'.
The default name is 'raw3'.

Your choice:

And "rawdj890c".

** creating printcap entry for printer rawdj890c...
  creating spooldir ...
  creating samba config file ...
  read protect password information...
  remember SETUP settings in printers apsfilterrc file...
** done.

[press <RETURN> to continue]

We're done, choose "Q" and hit "return". Read through the informational screens that follow. **apsfilter** has created the directories that are necessary and has modified the file /etc/printcap by adding the following information:

APS3_BEGIN:printer3
- don't delete start label for apsfilter printer3
- no other printer defines between BEGIN and END LABEL
dj890c|Printer3 auto:\
   :lp=/dev/null:\
   :if=/etc/apsfilter/basedir/bin/apsfilter:\

```
    :sd=/var/spool/lpd/dj890c:\
    :lf=/var/spool/lpd/dj890c/log:\
    :af=/var/spool/lpd/dj890c/acct:\
    :mx#0:\
    :sh:
rawdj890c|Printer3 raw:\
    :lp=/dev/null:\
    :if=/etc/apsfilter/basedir/bin/apsfilter:\
    :sd=/var/spool/lpd/rawdj890c:\
    :lf=/var/spool/lpd/rawdj890c/log:\
    :af=/var/spool/lpd/rawdj890c/acct:\
    :mx#0:\
    :sf:\
    :sh:
# APS3_END - don't delete this
```

Let's try it out with **lpr** by sending a postscript file to the printer. There is a very nice picture of a tiger's head that comes with **ghostcript**:

```
# lpr -Pdeskjet /usr/share/doc/gs/examples/tiger.ps.gz
```

Even a compressed postscript file gets printed nicely.

Setting up an nmbd WINS server

What is a WINS Server?

WINS stands for Windows Internet Name Service. This is a name service used to translate NetBIOS names to ip addresses by using NetBIOS over TCP/IP queries. It is done using UDP packets.

Using Samba as a WINS Server

To tell Samba that it should also play the role of WINS Server, add the following line to the [global] section of the Samba configuration file /etc/samba/smb.conf:

```
[global]
wins support = yes
```

Be careful, there should not be more than one WINS Server on a network and you should not set any of the other WINS parameters, such as "wins server", when enabling "wins support".

Using nmblookup to test the WINS Server

nmblookup is a Linux client that facilitates the lookup of NetBIOS names over TCP/IP.

Let's see if it works by asking **nmblookup** to find us the ip address for Yoda1:

```
pug:~# nmblookup Yoda1
querying Yoda1 on 192.168.2.255
192.168.2.21 Yoda1<00>
```

And let's prove that this is the same machine as Yoda:

```
pug:~# nmblookup Yoda
querying Yoda on 192.168.2.255
192.168.2.21 Yoda<00>
```

Another way to do this is with the **host** command:

```
pug:~# host 192.168.2.21
Name: yoda.falcon
Address: 192.168.2.21
```

To prove that yoda1 does not have a DNS entry:

```
pug:~# host yoda1
yoda1.falcon does not exist (Authoritative answer)
```

Another example: let's use **nmblookup** to find out which machine is the master browser for the falcon workgroup:

```
pug:~# nmblookup -M falcon
192.168.2.21 falcon<1d>
```

This proves that Yoda is the master browser for the falcon workgroup.

Creating logon scripts for clients

Logon scripts can be very handy. So for example, if every user needs his home directory mapped to drive H: automatically, a logon script can take care of that. The user is then presented with an extra hard-drive which gives you, as an administrator, the freedom to move home directories to another server should the need arise. To the user it remains drive H:, and all you have to do is

change one line in the logon script.

The same goes for printers and processes that should be accessible or run when a specific user logs on or when a certain machine logs on.

The batch file must be a Windows-style batch file and should thus have both a carriage return and a line feed at the end of each line.

The first thing to do is enable logon support. This is done by adding the following line to the [global] section of the Samba configuration file /etc/samba/smb.conf:

```
[global]
logon server = yes
```

The second thing to do is create a share called [netlogon] where the logon scripts will reside and which is readable to all users:

```
[netlogon]
  Comment = Netlogon for Windows clients
  path = /home/netlogon
  browseable = no
  guest ok = no
  writeable = no
```

The definition of the logon script depends on whether you want a script per user or per client.

Based on the user's name

Add the following line to the [netlogon] section:

```
logon script = %U.bat
```

and, assuming the user is "rc564", create a file called /home/netlogon/rc564.bat.

Based on the client's name

Add the following line to the [netlogon] section:

```
logon script = %m.bat
```

and, assuming the machine is called "xyz", create a file called /home/netlogon/xyz.bat.

Prev

Next

Implementing a Proxy Server (2.208.3) Home Configuring an NFS Server (2.209.2)

Configuring an NFS Server (2.209.2)

Author: Piet Plomp

Revision: $Revision: 1.9 $ ($Date: 2004/12/01 10:22:09 $)

Resources and further reading: NFS, NFSv4, Zadok01.

LPIC 2 objective 209.2

The candidate should be able to create an exports file and specify filesystems to be exported. This objective includes editing exports file entries to restrict access to certain hosts, subnets or netgroups. Also included is specifying mount options in the exports file, configuring user ID mapping, mounting an NFS filesystem on a client and using mount options to specify soft or hard and background retries, signal handling, locking and block size. The candidate should also be able to configure tcp wrappers to further secure NFS.

Key files, terms and utilities include:

The file /etc/exports
The exportfs command
The showmount command
The nfsstat command

NFS - The Network File System

The abbreviation NFS expands to *Network File System*. With NFS you can make a remote disk (or only some of it) part of your local filesystem.

The NFS protocol is currently being reworked, a process which has, so far, taken several years. This has consequences for those using NFS. Modern NFS daemons will currently run in *kernel space* (part of the running kernel) and support version 3 of the NFS protocol (version 2 will still be supported for compatibility with older clients). Older NFS daemons running in *user space* (which is almost independent of the kernel) and accepting only protocol version 2 NFS requests, will still be around. This section will primarily describe kernel-space NFS-servers supporting protocol version 3 and compatible clients. Differences from older versions will be pointed out when appropriate.

Note

Details about this NFS work-in-progress are in the section called "NFS protocol versions" below.

Client, Server or both?

The system that makes filesystem(s) available to other systems is called a *server*. The system that connects to a server is called a *client*. Each system can be configured as server, client or both.

Setting up NFS

This section describes NFS-related software and its configuration.

Requirements for NFS

To run NFS, the following is needed:

- support for NFS (several options) must be built into the *kernel*
- a *portmapper* must be running
- on systems with NFS-server support, an *NFS daemon* and a *mount daemon* must be active
- support daemons may be needed

Each is discussed in detail below.

Configuring the kernel for NFS

When configuring a kernel for NFS, it must be decided whether or not the system will be a client or a server. A system with a kernel that contains NFS-server support can also be used as an NFS client.

Note

The situation described here is valid for the 2.4.*x* kernel series. Specifications described here may change in the future.

NFS-related kernel options

NFS file system support (CONFIG_NFS_FS)

If you want to use NFS as a client, select this. If this is the only NFS option selected,

the system will support NFS protocol version 2 only. To use protocol version 3 you will also need to select CONFIG_NFS_V3. When CONFIG_NFS_FS is selected, support for an old-fashioned user-space NFS-*server* (protocol version 2) is also present. You can do without this option when the system is a kernel-space NFS-server server only (i.e., neither client nor user-space NFS-server).

Provide NFSv3 client support (CONFIG_NFS_V3)

Select this if the client system should be able to make NFS connections to an NFS version 3 server. This can only be selected if NFS support (CONFIG_NFS_FS) is also selected.

NFS server support (CONFIG_NFSD) Kernel space only.

When you select this, you get a *kernel-space* NFS-server supporting NFS protocol version 2. Additional software is needed to control the kernel-space NFS-server (as will be shown later). To run an old-fashioned user-space NFS-server this option is not needed. Select CONFIG_NFS instead.

Provide NFSv3 server support (CONFIG_NFSD_V3)

This option adds support for version 3 of the NFS protocol to the kernel-space NFS-server. The kernel-space NFS-server will support both version 2 and 3 of the NFS protocol. You can only select this if you also select NFS server support (CONFIG_NFSD).

When configuring during a compiler build (i.e., make menuconfig, make xconfig, etc), the options listed above can be found in the *File Systems* section, subsection *Network File Systems*.

Table 9.1, "Kernel options for NFS" provides an overview of NFS support in the kernel.

Table 9.1. Kernel options for NFS

Description	option(s)	allows / provides
NFS file system support	CONFIG_NFS_FS	allows both NFS (v2) client and user space NFS (v2) server
NFSv3 client support	CONFIG_NFS_FS and CONFIG_NFS_V3	allows NFS (v2 + v3) client
NFS server support	CONFIG_NFSD	provides NFS (v2) kernel server
NFSv3 server support	CONFIG_NFSD and CONFIG_NFSD_V3	provides NFS (v2 + v3) kernel server

Selecting at least one of the NFS kernel options turns on Sun RPC (Remote Procedure Call) support automatically. This results in a kernel space RPC input/output daemon. It can be recognised as [rpciod] in the process listing.

The portmapper

The portmapper is needed for all NFS traffic [16].

Most distributions will install the portmapper if NFS software (other than the kernel) is being installed.

The portmapper itself need not be configured. Portmapper security, however, *is* an issue: you are strongly advised to limit access to the portmapper. This can be done using the tcp wrapper.

Securing the portmapper

First, make sure the portmapper has support for the tcp wrapper built in. You can test this by running **ldd /sbin/portmap** [17]. The result could be something like

```
libwrap.so.0 => /lib/libwrap.so.0 (0x40018000)
libnsl.so.1 => /lib/libnsl.so.1 (0x40020000)
libc.so.6 => /lib/libc.so.6 (0x40036000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

The line with libwrap.so.0 (libwrap belongs to the tcp wrapper) shows that this portmapper is compiled with tcp-wrapper support. If that line is missing, get a better portmapper or compile one yourself.

A common security-strategy is blocking incoming portmapper requests by default, but allowing specific hosts to connect. This strategy will be described here.

Start by editing the file /etc/hosts.deny and adding the following line:

```
portmap: ALL
```

This denies every system access to the portmapper. It can be extended with a command:

```
portmap: ALL: (echo illegal rpc request from %h | mail root) &
```

Now all portmapper requests are denied. In the second example, requests are denied and

reported to root.

The next step is allowing only those systems access that *are* allowed to do so. This is done by putting a line in /etc/hosts.allow:

```
portmap: 121.122.123.
```

This allows each host with an IP address starting with the numbers shown to connect to the portmapper and, therefore, use NFS. Another possibility is specifying part of a hostname:

```
portmap: .example.com
```

This allows all hosts inside the example.com domain to connect. To allow all hosts of a NIS workgroup:

```
portmap: @workstations
```

To allow hosts with IP addresses in a subnet:

```
portmap: 192.168.24.16/255.255.255.248
```

This allows all hosts from 192.168.24.16 to 192.168.24.23 to connect (examples from [Zadok01]).

General NFS daemons

The nfs-utils package

NFS is implemented as a set of daemons. These can be recognized by their name: they all start with the *rpc.* prefix followed by the name of the daemon. Among these are: rpc.nfsd (only a support program in systems with a kernel NFS server), rpc.mountd, rpc.lockd and rpc.statd.

The source for these daemons can be found in the nfs-utils package (see [NFS] for more information on nfs-utils). It will also contain the source of other support programs, such as **exportfs**, **showmount** and **nfsstat**. These will be discussed later in the section called "Exporting filesystems" and the section called "Testing NFS".

Distributions may provide nfs-utils as a ready-to-use package, sometimes under different names. Debian, for example, provides lock and status daemons in a special nfs-common

package, and the NFS and mount daemons in nfs-*server packages (which come in user-space and kernel-space versions).

Each of the daemons mentioned here can also be secured using the tcp wrapper. Details in the section called "Securing NFS".

NFS server software

The NFS daemon

When implementing an NFS server, you can install support for an *kernel-space* or an *user-space* NFS server, depending on the kernel configuration. The **rpc.nfsd** command (sometimes called **nfsd**) **is** the complete NFS server in user space. I kernel space, however, it is just a support program that can start the NFS server in the kernel.

A kernel-space or a user-space NFS server

The kernel-space NFS server

The kernel-space NFS server is part of the running kernel. A kernel NFS server appears as [nfsd] in the process list.

The version of **rpc.nfsd** that supports the NFS server inside the kernel is just a support program to control NFS kernel server(s).

The user-space NFS daemon

The **rpc.nfsd** program can also contain an old-fashioned user-space NFS server (version 2 only). A user-space NFS server *is* a complete NFS server. It can be recognized as rpc.nfsd in the process list.

The mount daemon

The **mountd** (or **rpc.mountd**) mount-daemon handles incoming NFS (mount) requests. It is required on a system that provides NFS server support.

The configuration of the mount daemon includes *exporting* (making available) a filesystem to certain hosts and specifying how they can use this filesystem.

Exporting filesystems, using the /etc/exports file and the **exportfs** command will be discussed in the section called "Exporting filesystems".

The lock daemon

A lock daemon for NFS is implemented in **rpc.lockd**.

You won't need lock-daemon support when using modern (2.4.x) kernels. These kernels provide one internally, which can be recognized as [lockd] in the process list. Since the internal kernel lock-daemon takes precedence, starting **rpc.lockd** accidentally will do no harm.

There is no configuration for **rpc.lockd**.

The status daemon

According to the manual page, status daemon **rpc.statd** implements only a reboot notification service. It is a user-space daemon - even on systems with NFS version 3 support. It can be recognized as rpc.statd in the process listing. It is used on systems with NFS client and NFS server support.

There is no configuration for **rpc.statd**.

Exporting filesystems

Exporting a filesystem, or part of it, makes it available for use by another system. A filesystem can be exported to a single host, a group of hosts or to everyone.

Export definitions are configured in the file /etc/exports and will be activated by the **exportfs** command. The current export list can be queried with the command **showmount --exports**.

> **Note**
>
> In examples below the system called nfsshop will be the NFS server and the system called clientN one of the clients.

The file /etc/exports

The file /etc/exports contains the definition(s) of filesystem(s) to be exported, the name of the host that is allowed to access it and how the host can access it.

Each line in /etc/exports has the following format:

❶*/dir* ❷*hostname*(❸*options*) ❹...

❶ Name of the directory to be exported

❷ The name of the system (host) that is allowed to access */dir* (the exported directory). If the name of the system is omitted *all* hosts can connect. There are five possibilities to specify a system name:

single hostname

The name of a host that is allowed to connect. Can be a name (clientN) or an IP address.

wildcard

A group of systems is allowed. All systems in the example.com domain will be allowed by the *.example.com wildcard.

IP networks

Ranges of ip numbers or address/subnetmask combinations.

nothing

Leaving the system part empty is mostly done by accident (see the Caution below). It allows *all* hosts to connect. To prevent this error, make sure that there is no spacing between the system name and the opening brace that starts the options.

@NISgroup

NIS workgroups can be specified as a name starting with an @.

❸ Options between braces. Will be discussed further on.

❹ More than one system with options can be listed:

```
/home/ftp/pub clientN(rw) *.example.com(ro)
```

Explanation: system clientN is allowed to read and write in /home/ftp/pub. Systems in example.com are allowed to connect, but only to read.

Caution

Make sure there is *no space* (not even white) between the hostname and the specification between braces. There is a lot of difference between

```
/home/ftp/pub clientN(rw)
```

and

/home/ftp/pub clientN (rw)

The first allows clientN read and write access. The second allows clientN access with default options (see **man 5 exports**) and *all systems* read and write access!

Export options

Several export options can be used in /etc/exports. Only the most important will be discussed here. See the exports(5) manual page for a full list. Two types of options will be listed here: general options and user/group id options.

General options

ro (default)

The client(s) has only read access.

rw

The client(s) has read and write access. Of course, the client may choose to mount read-only anyway.

Also relevant is the way NFS handles user and group permissions across systems. NFS software considers users with the same UID and the same username as the same users. The same is true for GIDs.

The user root is different. Because root can read (and write) everything [18], root permission over NFS is considered dangerous.

A solution to this is called *squashing*: all requests are done as user nobody (actually UID 65534, often called -2) and group nobody (GID 65534).

At least four options are related to squashing: root_squash, no_root_squash, all_squash and no_all_squash. Each will be discussed in detail.

User/group id squashing

root_squash (default)

All requests by user root on clientN (the client) will be done as user nobody on nfsshop (the server). This implies, for instance, that user root on the client can only read files on the server that are *world readable*.

no_root_squash

All requests as root on the client will be done as root on the server.

This is necessary when, for instance, backups are to be made over NFS.

This implies that root on nfsshop completely trusts user root on clientN.

all_squash

Requests of any user other than root on clientN are performed as user nobody on nfsshop.

Use this if you cannot map usernames and UID's easily.

no_all_squash (default)

All requests of a non-root user on clientN are attempted as the same user on nfsshop.

Example entry in /etc/exports on system nfsshop (the server system):

```
/    client5(ro,no_root_squash) *.example.com(ro)
```

System nfsshop allows system clientN *read-only* access to everything and reads by user root are done as root on nfsshop. Systems from the example.com domain are allowed *read-only* access, but requests from root are done as user nobody, because root_squash is true by default.

Here is an example file:

```
# /etc/exports on nfsshop
# the access control list for filesystems which may be exported
# to NFS clients.  See exports(5).

/ client2.exworks(ro,root_squash)
/ client3.exworks(ro,root_squash)
/ client4.exworks(ro,root_squash)
/home client9.exworks(ro,root_squash)
```

Explanation: client2, client3 and client4 are allowed to mount the complete filesystem (/: root). But they have read-only access and requests are done as user nobody. The host client9 is only allowed to mount the /home directory with the same rights as the other three hosts.

The exportfs command

Once /etc/exports is configured, the export list in it can be activated using the **exportfs** command. It can also be used to reload the list after a change or deactivate the export list. Table 9.2, "Overview of exportfs" shows some of the functionality of **exportfs**.

Table 9.2. Overview of exportfs

Command	Description
exportfs -r	load or reload the export list
exportfs -ua	de-activate the export list

Note

Older (user-space) NFS systems may not have the **exportfs** command. On these systems the export list will be installed automatically by the mount daemon when it is started. Reloading after a change is done by sending a SIGHUP signal to the running mount-daemon process.

Activating an export list

The export list is activated (or reactivated) with the following command:

```
exportfs -r
```

The r originates from the word *re-exporting*.

Before the **exportfs -r** is issued, no filesystems are exported and no other system can connect.

When the export list is activated, the kernel export table will be filled. The following command will show the kernel export table:

```
cat /proc/fs/nfs/exports
```

The output will look something like:

```
# Version 1.1
# Path Client(Flags) # IPs
/   client4.exworks(ro,root_squash,async,wdelay) # 192.168.72.4
/home   client9.exworks(ro,root_squash,async,wdelay) # 192.168.72.9
/   client2.exworks(ro,root_squash,async,wdelay) # 192.168.72.2
/   client3.exworks(ro,root_squash,async,wdelay) # 192.168.72.3
```

Explanation: all named hosts are allowed to mount the root directory (client9: /home) of this machine with the listed options. The IP addresses are listed for convenience.

Also use **exportfs -r** after you have made changes to /etc/exports on a running system.

> **Warning**
>
> When running **exportfs -r**, some things will be done in the directory /var/lib/nfs. Files there are easy to corrupt by human intervention with far-reaching consequences, as I have learned from personal experience.

Deactivating an export list

All active export entries are unexported with the command:

```
exportfs -ua
```

The letters ua are an abbreviation for *unexport all*.

After the **exportfs -ua** no exports are active anymore.

The showmount command

The **showmount** shows information about the exported filesystems and active mounts to the host. Table 9.3, "Overview of showmount" shows how **showmount** can be used.

Table 9.3. Overview of showmount

Command	Description
showmount --exports	show active export list
showmount	show names of clients with active mounts
showmount --directories	show directories that are mounted by remote clients
showmount --all	show both client-names and directories

showmount accepts a host name as its last argument. If present, **showmount** will query the NFS-server on that host. If omitted, the current host will be queried (as in the examples below, where the current host is called nfsshop).

With the --exports **option.** The currently active export list can be queried with the --exports

option:

```
# showmount --exports
Export list for nfsshop:
/ client2.exworks,client3.exworks,client4.exworks
/home client9.exworks
```

The information is more sparse than the output of **cat /proc/fs/nfs/exports** shown earlier.

Without options. Without parameters, **showmount** will show names of hosts currently connected to the system:

```
# showmount
Hosts on nfsshop:
client9.exworks
```

With the --directories **option.** When the --directories option is specified, **showmount** will show names of directories that are currently mounted by a remote host:

```
# showmount --directories
Directories on nfsshop:
/home
```

With the --all **option.** The **showmount --all** command lists both the remote client (hosts) and the mounted directories:

```
# showmount --all
All mount points on nfsshop:
client9.exworks:/home
```

NFS client: software and configuration

An NFS *client* system is a system that does a mount-attempt, using the **mount** command. The **mount** needs to have support for NFS built-in. This will generally be the case.

The NFS client-system needs to have appropriate NFS support in the kernel, as shown earlier (see the section called "Configuring the kernel for NFS"). Next, it needs a running *portmapper*. Last, software is needed to perform the remote mounts attempt: the **mount** command.

Note

Familiarity with the **mount** command and the file /etc/fstab is assumed in this paragraph. If in doubt, consult the appropriate manual pages.

The **mount** command normally used to mount a remote filesystem through NFS:

mount -t nfs ❶*remote:/there* ❷*/here*

❶ This specifies the filesystem */there* on the remote server *remote*.

❷ The mount point */here* on the client, as usual.

Example: to mount the /usr filesystem, which is on server system nfsshop, onto the local mount-point /usr, use:

mount -t nfs nfsshop:/usr /usr

Fine-tuning of the mount request is done through *options*.

mount -t nfs ❶-o *opts* remote:/there /here

❶ Several options are possible after the -o option selector. These options affect either mount attempts or active NFS connections.

Mount options for NFS

ro versus rw

If ro is specified the remote NFS filesystem will be mounted *read-only*. With the rw option the remote filesystem will be made available for both reading and writing (if the NFS server agrees).

Tip

The default on the NFS server side (/etc/exports) is ro, but the default on the client side (**mount -t nfs**) is rw. The server-setting takes precedence, so mounts will be done *read-only*.

Tip

-o ro can also be written as -r; -o rw can also be written as -w.

rsize=*nnn* and wsize=*nnn*

The rsize option specifies the size for read transfers (from server to client). The wsize option specifies the opposite direction. A higher number makes data transfers faster on a reliable network. On a network where many retries are needed, transfers may become slower.

Default values are either 1024 or and 4096, depending on your kernel version. Current kernels accept a maximum of up to 8192. NFS version 3 over tcp, which will probably production-ready by the time you read this, allows a maximum size of 32768. This size is defined with NFSSVC_MAXBLKSIZE in the file include/linux/nfsd/const.h found in the kernel source-archive.

udp and tcp

Specifies the transport-layer protocol for the NFS connection. Most NFS version 2 implementations support only udp, but tcp implementations do exist. NFS version 3 will allow both udp and tcp (the latter is under active development). Future version 4 will allow only tcp. See the section called "NFS protocol versions".

nfsvers=*n*

Specifies the NFS version used for the transport (see the section called "NFS protocol versions"). Modern versions of **mount** will use version 3 by default. Older implementations that still use version 2 are probably numerous.

retry=*n*

The retry option specifies the number of minutes to keep on retrying mount-attempts before giving up. The default is 10000 minutes.

timeo=*n*

The timeo option specifies after how much time a mount-attempt times out. The time-out value is specified in deci-seconds (tenth of a second). The default is 7 deci-seconds (0.7 seconds).

hard (default) versus soft

These options control how hard the system will try.

hard. The system will try indefinitely.

soft. The system will try until an RPC (portmapper) timeout occurs.

intr versus nointr (default)

With these options one is able to control whether the user is allowed to interrupt the mount-attempt.

intr. A mount-attempt can be interrupted by the user if intr is specified.

nointr. A mount-attempt cannot be interrupted by a user if nointr is set. The mount request can seem to hang for days if retry has its default value (10000 minutes).

fg (default) and bg

These options control the *background mounting* facility. It is off by default.

bg. This turns on *background mounting*: the client first tries to mount in the foreground. All retries occur in the background.

fg. All attempts occur in the foreground.

Background mounting is also affected by other options. When intr is specified, the mount attempt will be interrupted by a an RPC timeout. This happens, for example, when either the remote host is down or the portmapper is not running. In a test setting the backgrounding was only done when a "connection refused" occurred.

Options can be combined using comma's:

```
mount -t nfs -o ro,rsize=8192 nfsshop:/usr/share /usr/local/share
```

A preferred combination of options might be: hard, intr and bg. The mount will be tried indefinitely, with retries in the background, but can still be interrupted by the user that started the mount.

Other mount options to consider are noatime, noauto, nosuid or even noexec. See **man 1 mount** and **man 5 nfs**.

Of course, all these options can also be specified in /etc/fstab. Be sure to specify the noauto option if the filesystem should **not** be mounted automatically at boot time. The user option will allow non-root users to perform the mount. This is not default. Example entry in /etc/fstab:

```
nfsshop:/home /homesOnShop  nfs ro,noauto,user   0  0
```

Now every user can do

```
mount /homesOnShop
```

You can also use automounters to mount and unmount remote filesystems, however, these are beyond the scope of this objective.

Testing NFS

After NFS has been set up, it can be tested. The following tools can help: **showmount**, **rpcinfo** and **nfsstat**.

The showmount --exports command

As shown in the section called "The **showmount** command", the **showmount --exports** command lists the current exports a server system. This can be used as a quick indication of the health of the created NFS system. There are more sophisticated ways.

rpcinfo

The **rpcinfo** command reports RPC information. This can be used to probe the portmapper on a local or a remote system or to send pseudo requests.

rpcinfo: probing a system

The **rpcinfo -p** command lists all registered services the portmapper knows about. Each *rpc...* program registers itself at startup with the portmapper, so the names shown correspond to real daemons (or the kernel equivalents, as is the case for NFS version 3).

It can be used on the server system nfsshop to see if the portmapper is functioning:

```
program vers proto   port
 100003   3   udp   2049  nfs
```

This selection of the output shows that this portmapper will accept connections for nfs version 3 on udp.

A full sample output of **rpcinfo -p** on a server system:

```
rpcinfo -p
program vers proto   port
 100000   2   tcp    111  portmapper
 100000   2   udp    111  portmapper
 100024   1   udp    757  status
 100024   1   tcp    759  status
 100003   2   udp   2049  nfs
 100003   3   udp   2049  nfs
 100021   1   udp  32770  nlockmgr
 100021   3   udp  32770  nlockmgr
 100021   4   udp  32770  nlockmgr
```

```
100005   1   udp  32771  mountd
100005   1   tcp  32768  mountd
100005   2   udp  32771  mountd
100005   2   tcp  32768  mountd
100005   3   udp  32771  mountd
100005   3   tcp  32768  mountd
```

As can be seen in the listing, the portmapper will accept RPC requests for versions 2 and 3 of the NFS protocol, both on udp.

Note

As can be seen, each RPC service has its own version number. The mountd service, for instance, supports incoming connections for versions 1, 2 or 3 of mountd on both udp and tcp.

It is also possible to probe nfsshop (the server system) from a client system, by specifying the name of the server system after **-p**:

```
rpcinfo -p nfsshop
```

The output, if all is well, of course, will be the same.

```
rpcinfo: making null requests
```

It is possible to test a connection without doing any real work:

rpcinfo -u *remotehost program*

This is like the **ping** command to test a network connection. However, **rpcinfo -u** works like a real rpc/nfs connection, sending a so-called *null* pseudo request. The **-u** option forces **rpcinfo** to use udp transport. The result of the test on nfsshop:

```
rpcinfo -u nfsshop nfs
program 100003 version 2 ready and waiting
program 100003 version 3 ready and waiting
```

The **-t** options will do the same for tcp transport:

```
rpcinfo -t nfsshop nfs
rpcinfo: RPC: Program not registered
program 100003 is not available
```

This system obviously does have support for nfs on udp, but not on tcp.

Note

In the example output, the number 100003 is used instead of or together with the name nfs. Name or number can be used in each others place. That is, we could also have written:

```
rpcinfo -u nfsshop 100003
```

The nfsstat command

The **nfsstat** lists statistics (i.e., counters) about nfs connections. This can be used to see whether something is going on at all and also to make sure nothing has gone crazy.

Table 9.4, "Some options for the nfsstat program" provides an overview of relevant options for **nfsstat**.

Table 9.4. Some options for the nfsstat program

	rpc	nfs	both
server	-sr	-sn	-s
client	-cr	-cn	-c
both	-r	-n	-nr

Sample output from **nfsstat -sn** on the server host nfsshop:

```
Server nfs v2:
null      getattr   setattr   root      lookup    readlink
1      0% 3       0% 0       0% 0       0% 41      0% 0       0%
read      wrcache   write     create    remove    rename
5595   99% 0      0% 0       0% 1       0% 0       0% 0       0%
link      symlink   mkdir     rmdir     readdir   fsstat
0      0% 0       0% 0       0% 0       0% 7       0% 2       0%

Server nfs v3:
null      getattr   setattr   lookup    access    readlink
1     100% 0      0% 0       0% 0       0% 0       0% 0       0%
read      write     create    mkdir     symlink   mknod
0      0% 0       0% 0       0% 0       0% 0       0% 0       0%
remove    rmdir     rename    link      readdir   readdirplus
```

```
0       0% 0       0% 0       0% 0       0% 0       0% 0       0%
fsstat     fsinfo     pathconf   commit
0       0% 0       0% 0       0% 0       0%
```

The 1's under both null headings are the result of the **rpcinfo -u nfsshop nfs** command shown earlier.

Securing NFS

NFS security has several unrelated issues. First, the NFS protocol and implementations have some known weaknesses. NFS file-handles are numbers that should be random, but are not, in reality. This opens the possibility of making a connection by guessing file-handles. Another problem is that all NFS data transfer is done as-is. This means that anyone able to listen to a connection can tap the information (this is called sniffing). Bad mount-point names combined with human error can be a totally different security risk.

Limiting access

Both sniffing and unwanted connection requests can be prevented by limiting access to each NFS server to a set of known, trusted hosts with trusted users on it: within a small workgroup, for instance. Tcp-wrapper support or firewall software can be used to limit access to an NFS server.

The tcp wrapper. Earlier (see the section called "Securing the portmapper") it was shown how to limit connections to the portmapper from specific hosts. The same can be done for the NFS related daemons, i.e., **rpc.mountd** and **rpc.statd**. If your system runs an old-fashioned user-space NFS server (i.e., has rpc.nfsd in the process list), consider protecting **rpc.nfsd** and possibly **rpc.lockd**, as well. If, on the other hand, your system is running a modern kernel-based NFS implementation (i.e., has [nfsd] in the process list), you cannot do this, since the **rpc.nfsd** program is not the one accepting the connections. Make sure tcp-wrapper support is built into each daemon you want to protect.

Firewall software. The problem with tcp-wrapper support is that there already is a connection inside the host when the connection is refused. If a security-related bug exists in either the tcp-wrapper library (not very likely) or the daemon that contains the support, unwanted access may be granted. Or worse. Firewall software (e.g., iptables) can make the kernel block connections before they enter the host. You can consider blocking unwanted NFS connections at each NFS server host or at the entry point of a network to all but acceptable hosts. Block at least the portmapper port (111/udp and 111/tcp). Also, considering blocking 2049/udp and 2049/tcp (NFS connections). You might also want to block other ports like the ones shown with the **rpcinfo -p** command: for example, the mount daemon ports 32771/udp and 32768/tcp (at least on my system). How to set up a firewall is shown in detail in Chapter 12, *System Security (2.212)*.

Preventing human error

Simple human error in combination with bad naming can also be a security risk. You would not be the first person to remove a remote directory tree because the mount point was not easily recognized as such and the remote system was mounted *read-write*.

Mount *read-only*. Mounting a remote filesystem *read-only* can prevent accidental erasure. So, mount read only if at all possible. If you do need to mount a part *read-write*, make the part that can be written (erased) as small as possible.

Design your mountpoints well. Also, name a mount point so that it can easily be recognized as a mount point. One of the possibilities is to use a special name:

```
/MountPoints/nfsshop
```

Best NFS version

Progress is made in NFS software. Although no software can prevent human error, other risks (e.g., guessable file-handles and sniffing) can be prevented with better software.

> **Note**
>
> NFS version 4 is a new version of the NFS protocol intended to fix all existing problems in NFS. It is still in the design phase. More about NFS version 4 and differences between the versions in the section called "NFS protocol versions" in a moment.

Guessable file handles. One of the ways to break in a NFS server is to guess so-called file-handles. The old (32-bit) file-handles (used in NFS version 2) were rather easy to guess. Version 3 of the NFS protocol offers improved security by using 64-bit file-handles that are considerably harder to guess.

Version 4 security enhancements. The upcoming version 4 of the NFS protocol defines encrypted connections. When the connection is encrypted, getting information by sniffing is made much harder or even impossible.

Overview of NFS components

Table 9.5, "Overview of NFS-related programs and files" provides an overview of the most important files and software related to NFS.

Table 9.5. Overview of NFS-related programs and files

program or file	description
The kernel	provides NFS support
The portmapper	handles RPC requests
rpc.nfsd	NFS server control (kernel space) or software (user space)
rpc.mountd	handles incoming (un)mount requests
The file /etc/exports	defines which filesystems are exported
The exportfs command	(un)exports filesystems
showmount --exports	shows current exports
The rpcinfo command	reports RPC information
The nfsstat command	reports NFS statistics
showmount --all	shows active mounts to me (this host)
mount -t nfs *remote:/there /here*	mounts a remote filesystem
umount -t nfs -a	unmounts all remote filesystems

NFS protocol versions

Currently, there are a lot of changes in the NFS protocol that can affect the way the system is set up. Table 9.6, "Overview of NFS protocol versions" provides an overview.

Table 9.6. Overview of NFS protocol versions

Protocol version	Current status	kernel or user space	udp or tcp transport
1	never released		
2	becoming obsolete	user, kernel	udp, some tcp impl. exist
3	new standard	kernel	udp, tcp: under development
4	future standard	kernel	tcp

The trends that can be seen in table Table 9.6, "Overview of NFS protocol versions" are: kernel space instead of user space and tcp instead of udp.

A note on the transport protocol. Connections over tcp (NFS v3,v4, some v2) are considered better than connections over udp (NFS v2,v3). The udp option might be the best on a small, fast network. But tcp allows considerably larger packet sizes (rsize, wsize) to be set. With sizes of 64k, tcp connections are reported to be 10% faster than connections over

udp, which does not allow sizes that large. See Zadok01 for a discussion about this.

---

[16] Strictly speaking, you can run NFS without a portmapper on a client system, however, connections will be slow and (even more) unreliable.

[17] The location of the portmapper may vary.

[18] Well, in most cases

Prev | Up | Next

Chapter 9. File and Service Sharing (2.209) | Home | Chapter 10. Network Client Management (2.210)

Chapter 10. Network Client Management (2.210)

Prev Next

Chapter 10. Network Client Management (2.210)

Revision: $Revision: 1.7 $ ($Date: 2004/03/03 15:03:36 $)

This topic has a total weight of 6 points and contains the following 4 objectives:

Objective 2.210.1; DHCP Configuration (2 points)

> This objective includes being able to configure a DHCP server, set default options, create a subnet, create a dynamically allocated range, adding a static host, setting options for a single host, adding bootp hosts, configure a DHCP relay client and reload the DHCP server after making changes.

Objective 2.210.2; NIS configuration (1 point)

> This objective includes being able to configure a NIS server, create NIS maps for major configuration files, configuring a system as a NIS client, setting up a NIS slave server and configuring the ability to search local files, DNS, NIS etc. in nsswitch.conf.

Objective 2.210.3; LDAP configuration (1 point)

> This objective includes being able to configure a LDAP server, configure a directory hierarchy, add groups, hosts, services and other data to the hierarchy, import items from LDIF files and add items with a management tool, as well as add users to the directory and change their passwords.

Objective 2.210.4; PAM authentication (2 points)

> This objective includes being able to configure PAM support authentication using /etc/passwd, shadow passwords, NIS and LDAP.

DHCP Configuration (2.210.1)

The candidate should be able to configure a DHCP server and set default options, create a subnet and create a dynamically-allocated range. This objective includes adding a static host, setting options for a single host and adding bootp hosts. Also included the configuration of a DHCP relay agent and reloading the DHCP server after making changes.

Key files, terms and utilities include:

dhcpd.conf
dhcpd.leases

What is DHCP?

DHCP stands for "Dynamic Host Configuration Protocol". DHCP consists of two components: a protocol for delivering client-specific configuration parameters from a DHCP server to a DHCP client and a mechanism for the allocation of network addresses to clients.

Amongst the most commonly used configuration items are: ip-address, host-name, domain-name, subnet-mask, broadcast-address, routers and domain-name-servers.

The information is requested by a DHCP client and provided by a DHCP server. By default, the server listens for requests on udp port 67 and answers through udp port 68, but it can be told to listen to another port instead with the -p option. The DHCP server will then answer through an udp port with a number one higher than the port it listens to.

The web-site *Resources for DHCP* contains a lot of (pointers to) information on the DHCP protocol, including RFC's.

How is the server configured?

The configuration of the DHCP server, **dhcpd**, is done by means of its configuration file /etc/dhcpd.conf.

The elements that can be used in a configuration file are: (global) parameters, shared networks, subnets, groups and hosts.

What are (global) parameters?

Parameters can be seen as variables that get assigned a value and are passed from the server to the client. Some parameters start with the *option* keyword and some do not. Parameters that do not start with the *option* keyword are either parameters that control the behavior of the DHCP server or are parameters that are optional in the DHCP protocol.

The difference between "normal" parameters and "global" parameters lies purely in the scope of the parameters. If, for instance, the DNS is always the same, it is pointless to add a domain-name-servers parameter-definition statement to every network-definition statement. By assigning the domain-name-servers parameter a value at the beginning of the configuration file, the parameter becomes a global parameter and its value becomes the default value for that parameter.

The value of a global parameter can be overridden by assigning the parameter another value in subsequent sections.

What is a shared-network declaration?

A shared-network declaration is used if there are multiple subnets on the same physical network. Parameters that are the same for all the subnets belonging to the shared-network can be defined once above the subnet-declarations within the shared-network declaration that encompasses those subnet-declarations.

What is a subnet declaration?

A subnet-declaration is used to define a network segment. Parameters that only apply to the subnet in question are defined within the subnet-declaration.

A subnet-declaration must contain a range statement that defines the IP-addresses the DHCP-server can give to clients on that subnet.

What is a group declaration?

A group-declaration is used to group other declarations, including group-declarations, that have a number of properties in common so that the common properties only have to be be specified once in stead of for every declaration.

What is a host declaration?

A host declaration is used to set properties for a specific client. The client identifies itself to the DHCP server by one of its unique properties such as its NIC address or its client-identifier.

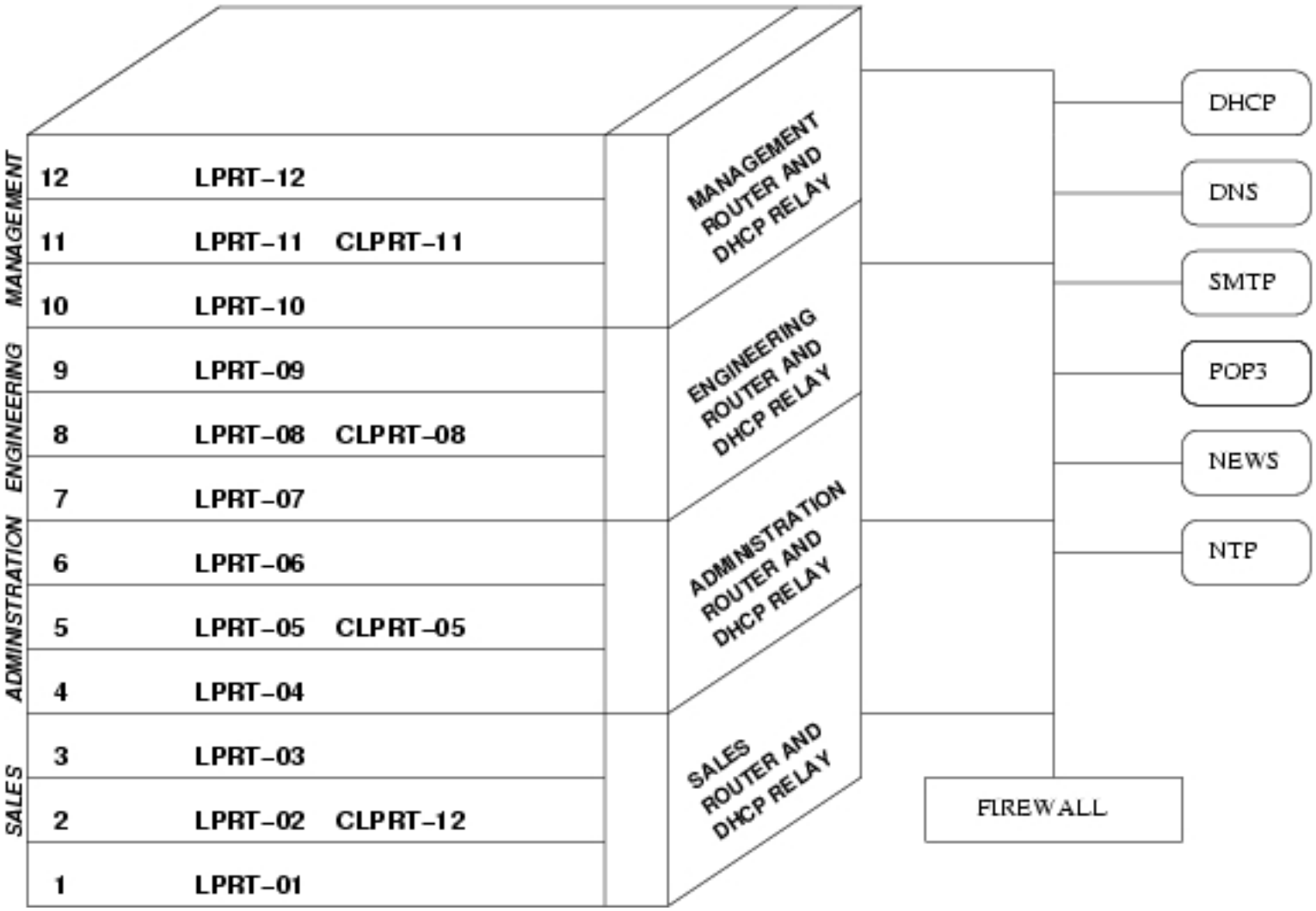
An example

Consider a firm which has four departments: Sales, Administration, Engineering and Management. All departments are located in the same building and each department has three floors to its disposal.

On each floor, there are up to 200 workstations and one laser printer (LPRT-xx). Furthermore each department has its own color laser-printer (CLPRT-xx) located on the middle floor. The printers can only be used by users of the department the printers belong to.

All users obtain an IP-address from the company's DHCP-server and must be able to reach the company's DNS-server and NTP-server. All users get their mail using the POP3 protocol, send their mail using the SMTP protocol and read their news using the NNTP protocol.

A graphical representation of the company's network is shown below:



The network architecture

Assuming that the IP range 21.31.x.x has been assigned to the company and that each department has its own network (determined by the highest four bits of the third octet), the subnets could be set up as follows:

Table 10.1. The first two octets are 21.31

Dept.	Floor	IP range	Router	Description
0001	0001	21.31.17.0 - 21.31.17.255		Sales floor #1
0001	0010	21.31.18.0 - 21.31.18.255	21.31.17.1	Sales floor #2
0001	0011	21.31.19.0 - 21.31.19.255		Sales floor #3
0010	0100	21.31.36.0 - 21.31.36.255		Administration #4

0010	0101	21.31.37.0 - 21.31.37.255	21.31.36.1	Administration #5
0010	0110	21.31.37.0 - 21.31.37.255		Administration #6
0011	0111	21.31.55.0 - 21.31.55.255		Engineering floor #7
0011	1000	21.31.56.0 - 21.31.56.255	21.31.55.1	Engineering floor #8
0011	1001	21.31.57.0 - 21.31.57.255		Engineering floor #9
0100	1010	21.31.74.0 - 21.31.74.255		Management floor #10
0100	1011	21.31.75.0 - 21.31.75.255	21.31.74.1	Management floor #11
0100	1100	21.31.76.0 - 21.31.76.255		Management floor #12

The network services available to workstations

The workstations on the company's network obtain their IP-address and the IP-addresses of the available network services from the company's DHCP-server via the department's DHCP-relay which also functions as a router.

Subnet-independent Services

Subnet-independent services are the services that are available to all workstations on the company's network regardless the subnet they are on. The table below shows those services and their fixed IP-addresses.

Table 10.2. Company-wide services

Service	Description	IP-address	Host name
DHCP	The company’s DHCP-server	21.31.0.1	dhcp.company.com
DNS	The company’s DNS	21.31.0.2	dns.company.com
SMTP	The company’s SMTP-server	21.31.0.3	smtp.company.com
POP3	The company’s POP3-server	21.31.0.4	pop3.company.com
NEWS	The company’s NNTP-server	21.31.0.5	news.company.com
NTP	The company’s NTP-server	21.31.0.6	ntp.company.com

Subnet dependent services

Subnet-dependent services are the services that are only available to the workstations on the same subnet as the service. The table below shows those services and their fixed IP-addresses.

Table 10.3. Subnet-dependent Services

Department	Service	Description	IP-address	Name
	Router	Sales Router floor #2	21.31.17.1	rtr-02.company.com
	Printer	Laser Printer floor #1	21.31.17.2	lprt-01.company.com
Sales	Printer	Laser Printer floor #2	21.31.18.2	lprt-02.company.com
	Printer	Laser Printer floor #3	21.31.19.2	lprt-03.company.com
	Printer	Color Laser Printer floor #2	21.31.18.3	clprt-02.company.com
	Router	Administration Router floor #5	21.31.36.1	rtr-05.company.com
	Printer	Laser Printer floor #4	21.31.36.2	lprt-04.company.com
Administration	Printer	Laser Printer floor #5	21.31.37.2	lprt-05.company.com
	Printer	Laser Printer floor #6	21.31.38.2	lprt-06.company.com
	Printer	Color Laser Printer floor #5	21.31.37.3	clprt-05.company.com
	Router	Engineering Router floor #8	21.31.55.1	rtr-08.company.com
	Printer	Laser Printer floor #7	21.31.55.2	lprt-07.company.com
Engineering	Printer	Laser Printer floor #8	21.31.56.2	lprt-08.company.com
	Printer	Laser Printer floor #9	21.31.57.2	lprt-09.company.com
	Printer	Color Laser Printer floor #8	21.31.56.3	clprt-08.company.com
	Router	Management Router floor #11	21.31.74.1	rtr-11.company.com
	Printer	Laser Printer floor #10	21.31.74.2	lprt-10.company.com
Management	Printer	Laser Printer floor #11	21.31.75.2	lprt-11.company.com
	Printer	Laser Printer floor #12	21.31.76.2	lprt-12.company.com
	Printer	Color Laser Printer floor #11	21.31.75.3	clprt-11.company.com

Building the DHCP-server's configuration file

The information needed to be able to build a configuration file has already been gathered in the previous sections when the network topology was devised.

In this section the actual configuration file /etc/dhcpd.conf will be filled with the necessary information.

The global parameters for services

Global parameters are put at the top of the configuration file:

```
# DNS
option domain-name-servers 21.31.0.2;
# SMTP
option smtp-server 21.31.0.3;
# POP3
option pop-server 21.31.0.4;
# NEWS
option nntp-server 21.31.0.5;
# NTP
option time-servers 21.31.0.6;
```

Another way to do this is by using domain names. A *single* domain name *must* resolve to a *single* IP-address. Using domain names, you would put the following entries in the configuration file:

```
# DNS
option domain-name-servers dns.company.com;
# SMTP
option smtp-server smtp.company.com;
# POP3
option pop-server pop3.company.com;
# NEWS
option nntp-server news.company.com;
# NTP
option time-servers ntp.company.com;
```

The company's shared-networks and subnets

As has been discussed in the previous sections, there are four different networks, one for each department and there are twelve different IP-address ranges, one for each floor. Furthermore, each network has its own router and printers.

This translates into four shared-networks each having their own netmask and broadcast-address and encompassing three IP-address ranges.

The netmask is an IP-address used to determine the network a workstation, or some other network device that uses an IP-address, is on. A netmask has 1's in the bit-positions that are the same for all network devices in that network and 0's in the other positions. Since all the subnets on a department's shared-network are on the same physical network, the distinction is made on the shared-network level, not on the floor level. The floor level has been coded into the IP-address (low-nibble of the third octet) to prepare for the planned installment next year of one router per floor in stead of one router per department. The

netmask is calculated as follows:

```
21.31.16.0 - : | 0001 0101 | 0001 1111 | 0001 0000 | 0000 0000 | SALES
21.31.31.255 : | 0001 0101 | 0001 1111 | 0001 1111 | 1111 1111 | NETWORK

21.31.32.0 - : | 0001 0101 | 0001 1111 | 0010 0000 | 0000 0000 | ADMINISTRATION
21.31.47.255 : | 0001 0101 | 0001 1111 | 0010 1111 | 1111 1111 | NETWORK

21.31.48.0 - : | 0001 0101 | 0001 1111 | 0011 0000 | 0000 0000 | ENGINEERING
21.31.63.255 : | 0001 0101 | 0001 1111 | 0011 1111 | 1111 1111 | NETWORK

21.31.64.0 - : | 0001 0101 | 0001 1111 | 0100 0000 | 0000 0000 | MANAGEMENT
21.31.79.255 : | 0001 0101 | 0001 1111 | 0100 1111 | 1111 1111 | NETWORK

fixed-bits   : | 1111 1111 | 1111 1111 | 1111 0000 | 0000 0000 | NETMASK
               255        255        240         0
```

Using a netmask of 255.255.240.0, the network an IP-address is on can be determined. This is done by AND-ing the IP-address with the netmask. To determine on which of the four networks a workstation with IP-address 21.31.57.105 is, the following calculation is performed:

```
21.31.57.105 : | 0001 0101 | 0001 1111 | 0011 1001 | 0110 1001 | IP-ADDRESS
255.255.240.0: | 1111 1111 | 1111 1111 | 1111 0000 | 0000 0000 | AND NETMASK
21.31.48.0:    | 0001 0101 | 0001 1111 | 0011 0000 | 0000 0000 | GIVES NETWORK
```

The IP-address 21.31.57.105 is on the 21.31.48.0 network, which is the Engineering-network.

The broadcast-address is used to send packets to every workstation on a network. A broadcast-address differs per network and can be determined by replacing all bits that can vary within a network with 1's.

Another way of determining the broadcast-address is to take the inverse of the netmask, in this case 0.0.15.255, and then OR the result with the network address:

```
21.31.16.0 - : | 0001 0101 | 0001 1111 | 0001 0000 | 0000 0000 | SALES
0.0.15.255   : | 0000 0000 | 0000 0000 | 0000 1111 | 1111 1111 | OR INV NETMASK
21.31.31.255 : | 0001 0101 | 0001 1111 | 0001 1111 | 1111 1111 | GIVES BCAST

21.31.32.0 - : | 0001 0101 | 0001 1111 | 0010 0000 | 0000 0000 | ADMINISTRATION
0.0.15.255   : | 0000 0000 | 0000 0000 | 0000 1111 | 1111 1111 | OR INV NETMASK
```

21.31.47.255 : | 0001 0101 | 0001 1111 | 0010 1111 | 1111 1111 | GIVES BCAST

21.31.48.0 - : | 0001 0101 | 0001 1111 | 0011 0000 | 0000 0000 | ENGINEERING
0.0.15.255 : | 0000 0000 | 0000 0000 | 0000 1111 | 1111 1111 | OR INV NETMASK
21.31.63.255 : | 0001 0101 | 0001 1111 | 0011 1111 | 1111 1111 | GIVES BCAST

21.31.64.0 - : | 0001 0101 | 0001 1111 | 0100 0000 | 0000 0000 | MANAGEMENT
0.0.15.255 : | 0000 0000 | 0000 0000 | 0000 1111 | 1111 1111 | OR INV NETMASK
21.31.79.255 : | 0001 0101 | 0001 1111 | 0100 1111 | 1111 1111 | GIVES BCAST

The broadcast-address for the network an IP-address is on can be determined by OR-ing the IP-address with the inverse-netmask. For a workstation with IP-address 21.31.57.105, the broadcast-address can be calculated as follows:

21.31.57.105 : | 0001 0101 | 0001 1111 | 0011 1001 | 0110 1001 | IP-ADDRESS
0.0.15.255 : | 0000 0000 | 0000 0000 | 0000 1111 | 1111 1111 | OR INV NETMASK
21.31.63.255 : | 0001 0101 | 0001 1111 | 0011 1111 | 1111 1111 | GIVES BCAST

The IP-address 21.31.57.105 belongs to a network that has broadcast-address 21.31.63.255, which is correct since the IP-address is on the Engineering-network.

To tell the DHCP-server what IP-addresses to give-out per subnet, a range stement must be added to the subnet. Is this example the IP-addresses 21.31.x.0 to 21.31.x.10 and 21.31.x.211 to 21.31.x.255 on every floor are reserved for printers and routers. This means that for every subnet the range statement is:

range 21.31.x.11 21.31.x.210

Where "x" depends on the department and the floor.

To implement this structure, the following lines are added to the configuration-file:

```
# The Sales network, floors 1-3
shared-network sales-net {
  # Sales-net specific parameters
  option routers 21.31.17.1;
  option lpr-servers 21.31.17.2, 21.31.18.2, 21.31.19.2, 21.31.18.3;
  option broadcast-address 21.31.31.255;
  subnet 21.31.17.0 netmask 255.255.240.0 {
    # Floor #1 specific parameters
    range 21.31.17.11 21.31.17.210;
```

```
  }
  subnet 21.31.18.0 netmask 255.255.240.0 {
    # Floor #2 specific parameters
    range 21.31.18.11 21.31.18.210;
  }
  subnet 21.31.19.0 netmask 255.255.240.0 {
    # Floor #3 specific parameters
    range 21.31.19.11 21.31.19.210;
  }
}

# The Administration network, floors 4-6
shared-network administration-net {
  # Administration-net specific parameters
  option routers 21.31.36.1;
  option lpr-servers 21.31.36.2, 21.31.37.2, 21.31.38.2, 21.31.37.3;
  option broadcast-address 21.31.47.255;
  subnet 21.31.36.0 netmask 255.255.240.0 {
    # Floor #4 specific parameters
    range 21.31.36.11 21.31.36.210;
  }
  subnet 21.31.18.0 netmask 255.255.240.0 {
    # Floor #5 specific parameters
    range 21.31.37.11 21.31.37.210;
  }
  subnet 21.31.19.0 netmask 255.255.240.0 {
    # Floor #6 specific parameters
    range 21.31.38.11 21.31.38.210;
  }
}

# The Engineering network, floors 7-9
shared-network engineering-net {
  # Engineering-net specific parameters
  option routers 21.31.55.1;
  option lpr-servers 21.31.55.2, 21.31.56.2, 21.31.57.2, 21.31.56.3;
  option broadcast-address 21.31.63.255;
  subnet 21.31.55.0 netmask 255.255.240.0 {
    # Floor #7 specific parameters
    range 21.31.55.11 21.31.55.210;
  }
  subnet 21.31.18.0 netmask 255.255.240.0 {
    # Floor #8 specific parameters
```

```
   range 21.31.56.11 21.31.56.210;
  }
  subnet 21.31.19.0 netmask 255.255.240.0 {
   # Floor #9 specific parameters
   range 21.31.57.11 21.31.57.210;
  }
}

# The Management network, floors 10-12
shared-network management-net {
  # Management-net specific parameters
  option routers 21.31.74.1;
  option lpr-servers 21.31.74.2, 21.31.75.2, 21.31.76.2, 21.31.75.3;
  option broadcast-address 21.31.79.255;
  subnet 21.31.74.0 netmask 255.255.240.0 {
   # Floor #10 specific parameters
   range 21.31.74.11 21.31.74.210;
  }
  subnet 21.31.18.0 netmask 255.255.240.0 {
   # Floor #11 specific parameters
   range 21.31.75.11 21.31.75.210;
  }
  subnet 21.31.19.0 netmask 255.255.240.0 {
   # Floor #12 specific parameters
   range 21.31.76.11 21.31.76.210;
  }
}
```

Static hosts

A static host is a host that always gets the same IP-address from the DHCP-server in opposite to dynamic hosts which get their IP-address from a range of IP-addresses.

Obviously, the DHCP-server must be able recognize the host to be able to conclude that the host has been defined as a static one in the DHCP-server's configuration file. This can be done by using the dhcp-client-identifier option or by using the hardware ethernet option.

The dhcp-client-identifier is send to the server by the client (host) and must uniquely identify that client. This is not safe because there is no way to be sure that there isn't a second client that uses the same identifier.

The hardware ethernet option causes the match to be done on the client's NIC-address which is world-wide unique.

If the client does not send a dhcp-client-identifier, then the NIC-address is used to identify the client.

There are two designers, working for the Engineering department, that come to the office sometimes to get a hardcopy of their designs in colour. These designers are called "luke" and "leah" and they bring their laptops and connect them to the Engineering-network. The host names of their machines will be "luke" and "leah".

To make this so, the administrator has added the following lines to the DHCP-server's configuration file:

```
group {
    # options that apply to all the static hosts
    option routers 21.31.55.1;
    option lpr-servers 21.31.56.3;
    option broadcast-address 21.31.63.255;
    netmask 255.255.240.0;
    host luke {
        # specific for luke
        hardware ethernet AA:88:54:72:7F:92;
        fixed-address 21.31.55.211;
        option host-name "luke";
    }

    host leah {
        # specific for leah
        hardware ethernet CC:88:54:72:84:4F;
        fixed-address 21.31.55.212;
        option host-name "leah";
    }
}
```

Static BOOTP hosts

This is a special static host. If luke and leah's laptops were BOOTP-clients, the administrator could have added the following lines to the DHCP-server's configuration file:

```
group {
    # options that apply to all the static hosts
    option routers 21.31.55.1;
    option lpr-servers 21.31.56.3;
```

```
    option broadcast-address 21.31.63.255;
    netmask 255.255.240.0;
    host luke {
          # specific for luke
          filename "lukes-boot-file";
          server-name "server name to send to luke";
          next-server <address of server to load boot-file from>;
          hardware ethernet AA:88:54:72:7F:92;
          fixed-address 21.31.55.211;
          option host-name "luke";
    }

    host leah {
          # specific for leah
          filename "leahs-boot-file";
          server-name "server name to send to leah";
          next-server <address of server to load boot-file from>;
          hardware ethernet CC:88:54:72:84:4F;
          fixed-address 21.31.55.212;
          option host-name "leah";
    }
}
```

The filename option states the name of the file to get from the server defined in the next-server option. If the next-server is omitted, the server to get the file from is the DHCP-server. The server-name can be used to send the name of the server the client is booting from to the client.

For information on the valid options, consult the dhcp-options man page (**man dhcp-options**) and the dhcpd.conf man page (**man dhcpd.conf**).

Controlling the DHCP-server's behavior

Leases

A lease is the amount of time a client may use the IP-address it got from the DHCP-server. The client must refresh the lease periodically because the IP-address can be given to another client if the lease is expired. Normally, a client will be given the same IP-address if the lease is refreshed before it expired.

The option max-lease-time is used to specify the maximum amount of time in seconds that will be assigned to a lease if the client asks for a specific expiration time.

The option default-lease-time is used to specify the amount of time in seconds that will be assigned to a lease if a client does not ask for a specific expiration time.

The DHCP-server keeps a database of the leases it has issued in the file /var/dhcp/dhcpd.leases. If this file is empty, this is probably caused by the fact that you have only defined static hosts in the DHCP-server's configuration file and you haven't used any **range** statements.

Interfaces the DHCP-server listens on

Unless you specify otherwise, **dhcpd** will listen on *all interfaces* for a dhcp request. If you only want to serve requests on *eth0* for instance, you can tell this to the daemon by including the parameter on the command line that starts the daemon.

Reloading the DHCP-server after making changes

This is done as follows:

```
# /etc/init.d/dhcp restart
```

This will stop the running daemon, wait two seconds, then start a new daemon which causes /etc/dhcpd.conf to be read again.

DHCP-relaying

What is DHCP-relaying?

In our earlier example there is one DHCP-server for the whole network and there are routers between the clients and that server.

If a client would be able to connect to the DHCP-server through a router, the DHCP-server would not see the NIC-address of the client but the NIC-address of the router. This would mean that a static host for instance, could not be recognized by its hardware address.

A DHCP-relay agent, such as **dhcrelay** provides a means for relaying DHCP and BOOTP requests from one of the subnets to the company's DHCP-server.

If you need more information, consult *The Internet Consortium DHCP Homepage*.

If not specified otherwise on the command line, the DHCP-relay agent listens for DHCP requests on all interfaces and forwards them to the DHCP-servers specified on the command line. A reply is only sent to the network the request came from.

Please consult the man page (**man dhcrelay**) for further details.

Prev | Next

Configuring an NFS Server (2.209.2) | Home | NIS configuration (2.210.2)

NIS configuration (2.210.2)

NIS configuration (2.210.2)

The candidate should be able to configure an NIS server and create NIS maps for major configuration files. This objective includes configuring a system as a NIS client, setting up an NIS slave server and configuring the ability to search local files, DNS, NIS, etc. in nsswitch.conf.

Key files, terms and utilities include:

nisupdate, ypbind, ypcat, ypmatch, ypserv, ypswitch, yppasswd, yppoll, yppush, ypwhich, rpcinfo

nis.conf, nsswitch.conf, ypserv.conf

Contents of /etc/nis/: netgroup, nicknames, securenets

Makefile

What is it?

NIS stands for Network Information System. NIS was developed by Sun Microsystems and was originally named "Yellow Pages" but since that is a trademark of British Telecom, Sun Microsystems had to change that name and choose NIS. As a remnant of that former name, all NIS commands start with the letters "yp".

NIS is based on RPC (Remote Procedure Calls) and consists of a server and one or more clients. Because NIS uses RPC calls, the portmapper (**portmap**) *must* be running. See the section called "Configuring an NFS Server (2.209.2)" for more information on the portmapper.

The purpose of NIS is to distribute information on the server to all hosts on the network, thus keeping the network transparent to the users. Let's say, for example, that you would like users to be able to log into any system on the LAN using the same userid and password.

This can be achieved by telling NIS to distribute the files /etc/passwd and /etc/groups to all hosts in the NIS domain.

The files distributed across the network can be any type of file, e.g., a text file that contains the phone numbers of your colleagues.

Configuring a system as a NIS client

The NIS domain name, which is case-sensitive, must be set. This is done either during the installation (when you are prompted for the domainname) or afterwards by editing the file /etc/defaultdomain.

Normally a client finds its NIS servers by broadcasting on the local network. If the NIS server is not on that network, you must specify the server(s) in /etc/yp.conf.

Then NIS can be started with the command **/etc/init.d/nis start**.

Setting up NIS master and slave servers

What is their relation?

A NIS master-server uses the command **yppush** to copy its NIS databases to one or more slave servers. What actually happens is that the NIS master-server tells a slave to come and get the databases. The slave then uses **ypxfr** to do that.

A drawback is that this only works if the slave happens to be up and running at the moment that the server executes **yppush**. The **ypxfr** manual page suggests that you run **ypxfr** periodically in a cron job and make the frequency dependent on the mutation grade of the maps. Please consult this man page for further details (**man 8 ypxfr**).

Configuring them

There should be a HOWTO for your distribution that describes this. On my Debian system, I found the information in the file /usr/share/doc/nis.debian.howto.gz.

Set the NIS domain in the /etc/defaultdomain file. It is common practice to use your DNS domain name for this.

The next thing to do is restrict access to your NIS domain because everyone who knows your domain name can retrieve the contents of your NIS maps. There are two files where access restrictions can be configured: /etc/ypserv.conf and /etc/ypserv.securenets.

If none of the access rules in /etc/ypserv.conf matches the map name, **ypserv** checks if there is an YP_SECURE key in the map. If so, access is allowed on a reserved port, if not, access is allowed. An access rule has the following format, please read the man page for further details (**man ypserv.conf**):

host:map:security:mangle[:field]

This is not very safe because you have to specify what is not allowed instead of what is. It is

safer to specify from which hosts access is allowed. This can be done in one of two ways, either with tcpwrappers which involves the files /etc/hosts.allow and /etc/hosts.deny or with the securenets method of **ypserv** which involves the file /etc/ypserv.securenets.

Each line in the /etc/ypserv.securenets consists of a netmask and a network, for instance:

```
255.255.255.0 101.102.103.0
```

This means that all machines in the 101.102.103.0 network are allowed to connect to the NIS server. If none of the rules match an incoming request, the request is logged and ignored. If, however, the file /etc/ypserv.securenets does not exist, connections from all host are allowed.

To use tcpwrapper support, this functionality needs to have been compiled into **ypserv**. If this is the case, tcpwrappers is used instead of the securenets functionality. To test if this is the case, type:

```
# ypserv --version
ypserv - NYS YP Server version 1.3.12 (with securenets)
```

Obviously, I have not compiled tcpwrappers into **ypserv**.

Creating NIS maps

NIS maps are created by running **make** in the directory /var/yp/.

make reads the file /var/yp/Makefile which contains the definitions of the NIS environment and the various maps.

The file /var/yp/Makefile explains itself, I suggest you read through it at least once.

NIS related commands and files

This section briefly describes the functionality of the yp* commands and related files as described in their man pages. Please consult those man pages for further details on command line options and other features.

The related yp* commands and file:

ypbind

ypbind finds the server for NIS domains and maintains the NIS binding information.

ypcat

ypcat prints the values of all keys from the NIS database specified by mapname, which may be a map name or a map nickname.

ypchfn, ypchsh and yppasswd

The standard **passwd**, **chfn** and **chsh** cannot be used under to change the users' NIS password, shell or GECOS information, because only the password file on the local host is modified. For changing the NIS information, they are replaced by their NIS counterparts: **yppasswd**, **ypchfn** and **ypchsh**.

ypdomainname

ypdomainname sets or displays the name of the current NIS domain.

ypmatch

ypmatch prints the values of one or more keys from the NIS database specified by mapname.

yppoll

yppoll returns the version and master server of a NIS map.

yppush

yppush forces the propagation of changed NIS databases.

ypserv

This is the Network Information Service (NIS) server daemon.

ypset

ypset binds **ypbind** to a specific NIS server.

ypwhich

ypwhich returns the name of the NIS server or the name of the master for a map.

nisupdate

All I could find about **nisupdate** is that it is a script that is part of the Webmin suite. Webmin is a web-based interface for system administration for Unix.

If you would like to know more about Webmin, go to the *Webmin Website*.

nis.conf

This seems to be distribution specific. The file is not present on my Debian system. The file /etc/defaults/nis contains the Configuration settings for the NIS daemons as shown below:

```
#
# /etc/defaults/nis Configuration settings for the NIS daemons.
#

# Are we a NIS server and if so what kind (values: false, slave, master)
NISSERVER=false

# Location of the master NIS password file (for yppasswdd).
# If you change this make sure it matches with /var/yp/Makefile.
YPPWDDIR=/etc

# Do we allow the user to use ypchsh and/or ypchfn ? The YPCHANGEOK
# fields are passed with -e to yppasswdd, see it's manpage.
# Possible values: "chsh", "chfn", "chsh,chfn"
YPCHANGEOK=chsh
```

nsswitch.conf

/etc/nsswitch.conf is the System Databases and Name Service Switch configuration file. Various functions in the C Library need to be configured to work correctly in the local environment. Traditionally, this was done by using files (e.g. /etc/passwd), but other nameservices (like the Network Information Service (NIS) and the Domain Name Service (DNS)) became popular, and were hacked into the C library, usually with a fixed search order. The Linux GNU C Library 2.x (libc.so.6) contains a cleaner solution of this problem. It's design is based upon a method used by Sun Microsystems in the C library of Solaris 2. Sun calls this scheme "Name Service Switch" (NSS). The sources for the "databases" and their lookup order are specified in the /etc/nsswitch.conf file.

The file /etc/nsswitch.conf consists of one line per database. The first item on the line is the name of the database followed by a colon (:). The rest of the line specifies how the lookup is done and how lookup results are treated. For example:

```
hosts:      dns [!UNAVAIL=return] files
networks:   nis [NOTFOUND=return] files
```

The available databases are: aliases, ethers, group, hosts, netgroup, network, passwd, protocols, publickey, rpc, services, shadow

Amongst the services are: compat, db, files, hesiod, nis, nisplus For every service there must be a file /lib/libnss_<service>.so.<version>.

For glibc 2.0 the version number is "1", for glibc 2.1 the version number is "2".

The action items are placed within brackets and between two service names. The general form is:

[(!STATUS = ACTION)+] where

'!' negates the STATUS and can be omitted
STATUS can be: success, notfound, unavail or tryagain
ACTION can be: return or continue

Please consult the man page (**man nsswitch.conf**) for further details.

ypserv.conf

/etc/ypserv.conf is the configuration file for **ypserv** and **rpc.ypxfrd**.

Prev | Up | Next

Chapter 10. Network Client Management (2.210) | Home | LDAP configuration (2.210.3)

LDAP configuration (2.210.3)

The candidate should be able to configure an LDAP server. This objective includes configuring a directory hierarchy and adding group, hosts, services and other data to the hierarchy. Also included are: importing items from LDIF files and adding items with a management tool, as well as adding users to the directory and changing their passwords.

Key files, terms and utilities include:

slapd

slapd.conf

What is it?

LDAP stands for Lightweight Directory Access Protocol. As the name suggests, it is a lighter version of DAP, which stands for the Directory Access Protocol that is defined by the X.500 standard. For more information on X.500, please read *RFC 2116*.

The reason for a lightweight version is that DAP was rather heavy on processor load, thereby asking for more than the processors could provide at the time. LDAP is described in *RFC 2251*.

The LDAP project was started at the *University of Michigan*, but, as can be read on their site, is no longer maintained there. For current information, the University of Michigan site points visitors to the *OpenLDAP* site instead.

The type of information best suited for storage in a directory is information with a low mutation grade. The reason for this is that directories can not compete with RDBM systems because they are only optimized for read access. So then, what do we store in a directory? Typically, LDAP directories contain employee data such as surname, christian name, address, phone number, department, social security number, E-mail address. Alternatively, directories might store newsletters for everyone to read, description of company policies and procedures, templates supporting the house style of documents.

Installing and Configuring an LDAP Server

Obtaining the software

The Open Source LDAP software can be downloaded from the *OpenLDAP Download* site.

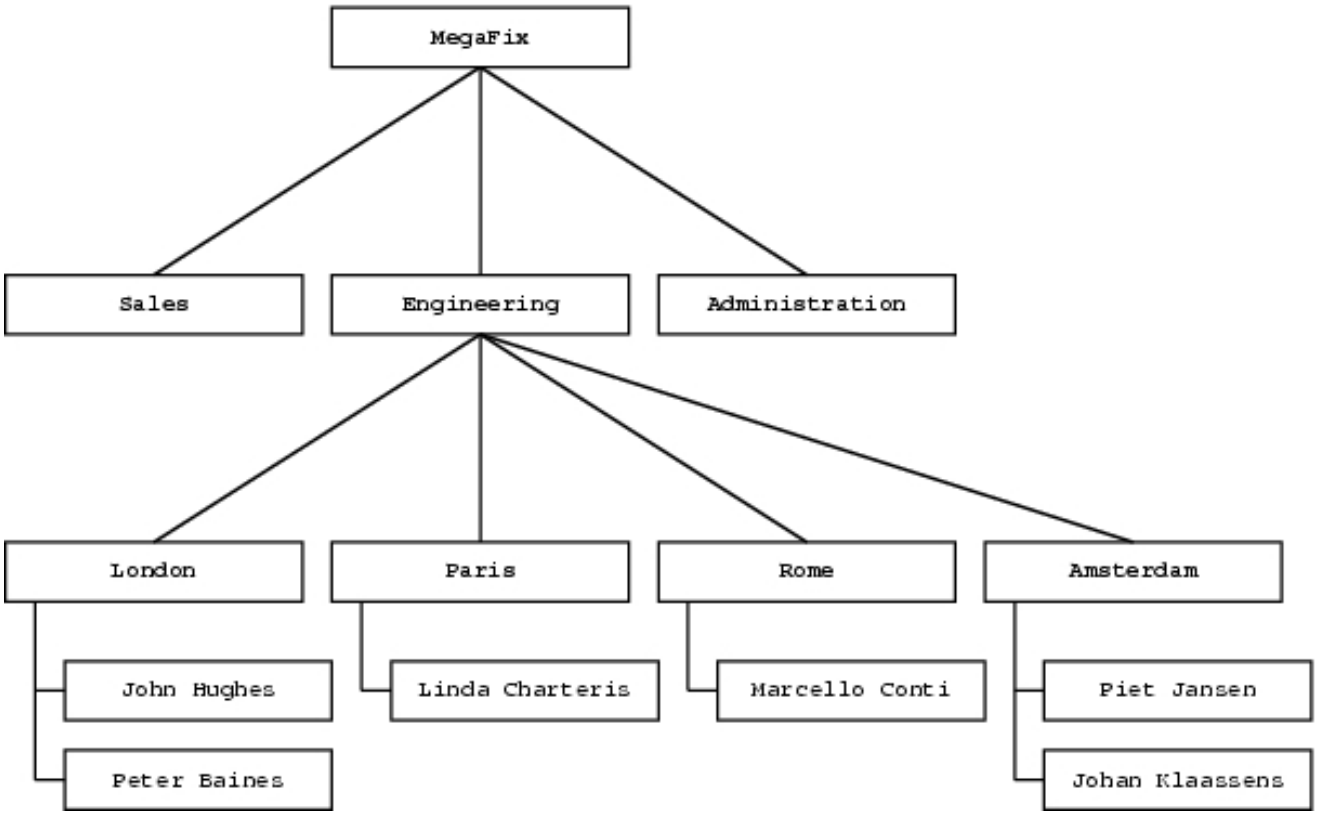
Detailed instructions on installing LDAP can be found in the *OpenLDAP Administrator's Guide*.

On a Debian system, **apt-get install slapd** will do the trick. You will have to edit the file /etc/ldap/slapd.conf to reflect your wishes.

After this, the **slapd** daemon can be started with the command **/etc/init.d/slapd start**.

Configuring a directory hierarchy

Let's take the company MegaFix in The Netherlands as an example.



This figure shows MegaFix's organisational structure.

The company consists of three departments: Sales, Engineering and Administration.

In order to provide local support to their customers, the Engineering department has offices in London, Paris, Rome and Amsterdam with, for the moment, one or two engineers.

So, how do we build a directory for this organization?

First, we must decide what kind of information to store in the directory. Let's put ourselves in the shoes of the customer:

i. The organizational structure of the company
ii. Information about people working at the company

The second thing to do is select or create object class and schema definitions. The directory /etc/ldap/schema contains a number of predefined ones and core.schema fits our needs (trust me, or have a look at it if you don't).

The third thing to do is choose the object class we are going to use. This depends on the data we wish to store. An object class describes an object by defining its possible attributes.

For the organizational structure of the company, we are going to use the object classes **organization**, **organizationalUnit** (note the "z") and **country**. This gives us the following attributes:

- country or "c" in short notation
- organization or "o" in short notation
- organizationalUnit or "ou" in short notation
- userPassword

- searchGuide
- seeAlso
- businessCategory
- x121Address
- registeredAddress
- destinationIndicator
- preferredDeliveryMethod
- telexNumber
- teletexTerminalIdentifier
- telephoneNumber
- internationaliSDNNumber
- facsimileTelephoneNumber
- street
- postOfficeBox
- postalCode
- postalAddress
- physicalDeliveryOfficeName
- stateOrProvinceName or "st"
- localityName or "l"
- description

For people working for the company, we are going to use the object class **person**, which supports the following attributes:

- commonName or "cn" in short notation
- surname or "sn" in short notation
- userPassword
- telephoneNumber
- seeAlso
- description

Let's configure **slapd** to reflect these choices by editing the configuration file /etc/ldap/slapd.conf:

```
# Schema and objectClass definitions
include     /etc/ldap/schema/core.schema

# Schema check allows for forcing entries to
# match schemas for their objectClasses's
schemacheck off

# Where the pid file is put. The init.d script
# will not stop the server if you change this.
pidfile     /var/run/slapd.pid

# List of arguments that were passed to the server
argsfile    /var/run/slapd.args

# Read slapd.conf(5) for possible values
loglevel    0

########################################################################
# ldbm database definitions
########################################################################

# The backend type, ldbm, is the default standard
database    ldbm
```

```
# The base of your directory
suffix      "o=MegaFix,c=NL"

# Where the database files are physically stored
directory   "/var/ldap-test"

# Distinguished name, not subject to access control
rootdn "cn=Manager,o=MegaFix,c=NL"
rootpw blabla

# Save the time that the entry gets modified
lastmod on
```

For further details on the configuration file, please read the man page (**man slapd.conf**).

Nest, we need to create the directory where the databases will reside: **mkdir /var/ldap-test**. Now we're ready to start the daemon:

```
# /etc/init.d/slapd start
Starting ldap server(s): slapd.
```

It seems to work. Let's see if there are any error messages that have been logged:

```
# ls -ltr /var/log
...
-rw-r-----   1 root     adm       934047 Jan 25 16:44 syslog
-rw-r-----   1 root     adm        45275 Jan 25 16:44 debug
-rw-r-----   1 root     adm       282043 Jan 25 16:45 auth.log
```

There are no messages in syslog and debug but auth.log contains:

```
# grep "slapd" /var/log/auth.log
Jan 25 16:44:22 pug slapd[1373]: unable to open Berkeley db /etc/sasldb: No
such file or directory
```

SASL stands for "Simple Authentication and Security Layer". This is the layer between OpenLDAP and Kerberos. Since we're not using Kerberos, let's silently ignore the message.

Finally, we define the MegaFix organization in LDAP Data Interchange Format (ldif). See ldif's man page (**man ldif**) for further details.

```
#---------------------------------------------------------
# The organisational structure
#
# dn = distinguishedName
# ou = organizationalUnit
# o  = organizationName
# c  = country
#---------------------------------------------------------
# The organisation MegaFix in the Netherlands
```

```
dn: o=MegaFix, c=NL
objectClass: organization
description: The MegaFix Company Ltd.

# The Sales department
dn: ou=Sales, o=MegaFix, c=NL
objectClass: organization
description: Sales dept.

# The engineering department
dn: ou=Engineering, o=MegaFix, c=NL
objectClass: organization
description: Engineering dept.

# Engineering - London division
dn: ou=London, ou=Engineering, o=MegaFix, c=NL
objectClass: organization
description: Division London

# Engineering - Paris division
dn: ou=Paris, ou=Engineering, o=MegaFix, c=NL
objectClass: organization
description: Division Paris

# Engineering - Rome division
dn: ou=Rome, ou=Engineering, o=MegaFix, c=NL
objectClass: organization
description: Division Rome

# Engineering - Amsterdam division
dn: ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
objectClass: organization
description: Division Amsterdam

dn: ou=Administration, o=MegaFix, c=NL
objectClass: organization
description:  Administration dept.

#-------------------------------------------------------------
# The persons in the organisation
#
# dn = distinguishedName
# ou = organizationalUnit
# o  = organizationName
# c  = country
# cn = commonName
# sn = surname
#-------------------------------------------------------------

# The Company's Manager
dn: cn=Manager, o=MegaFix, c=NL
objectClass: person
cn: Manager
cn: Gordon Gekko
```

```
sn: Gekko
description: General Manager - The Big Boss
telephoneNumber: 555-1255

# The engineers in London
dn: cn=John Hughes, ou=London, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: John Hughes
sn: Hughes
description: Engineer

dn: cn=Peter Baines, ou=London, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: Peter Baines
sn: Baines
description: Engineer

# The engineers in Paris
dn: cn=Linda Charteris, ou=Paris, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: Linda Charteris
sn: Charteris
description: Engineer

# The engineers in Rome
dn: cn=Marcello Conti, ou=Rome, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: Marcello Conti
sn: Conti
description: Engineer

# The engineers in Amsterdam
dn: cn=Pieter Jansen, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: Pieter Jansen
sn: Jansen
description: Engineer

dn: cn=Johan Klaassens, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: Johan Klaassens
sn: Klaassens
description: Engineer
```

To update our directory with the data from the above file MegaFix.ldif, we use the command:

```
# ldapadd -f /etc/ldap/MegaFix.ldif -D 'cn=Manager,o=Megafix,c=NL' -W -x
```

```
Enter LDAP Password: (which is 'blabla')
adding new entry "o=MegaFix, c=NL"
adding new entry "ou=Sales, o=MegaFix, c=NL"
adding new entry "ou=Engineering, o=MegaFix, c=NL"
adding new entry "ou=London, ou=Engineering, o=MegaFix, c=NL"
adding new entry "ou=Paris, ou=Engineering, o=MegaFix, c=NL"
adding new entry "ou=Rome, ou=Engineering, o=MegaFix, c=NL"
adding new entry "ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL"
adding new entry "ou=Administration, o=MegaFix, c=NL"
adding new entry "cn=Manager, o=MegaFix, c=NL"
adding new entry "cn=John Hughes, ou=London, ou=Engineering, o=MegaFix, c=NL"
adding new entry "cn=Peter Baines, ou=London, ou=Engineering, o=MegaFix, c=NL"
adding new entry "cn=Linda Charteris, ou=Paris, ou=Engineering, o=MegaFix, c=NL"
adding new entry "cn=Marcello Conti, ou=Rome, ou=Engineering, o=MegaFix, c=NL"
adding new entry "cn=Pieter Jansen, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL"
adding new entry "cn=Johan Klaassens, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL"
```

The data has been added.

To check if everything works the way we want it to, let's ask our directory for a list of all entries concerning the "engineering" organizationalUnit, "ou" for short:

```
# ldapsearch -LLL -b 'ou=engineering,o=MegaFix,c=nl' -x cn description
dn: ou=Engineering, o=MegaFix, c=NL
description: Engineering dept.

dn: ou=London, ou=Engineering, o=MegaFix, c=NL
description: Division London

dn: ou=Paris, ou=Engineering, o=MegaFix, c=NL
description: Division Paris

dn: ou=Rome, ou=Engineering, o=MegaFix, c=NL
description: Division Rome

dn: ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
description: Division Amsterdam

dn: cn=John Hughes, ou=London, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: John Hughes
description: Engineer

dn: cn=Peter Baines, ou=London, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: Peter Baines
description: Engineer

dn: cn=Linda Charteris, ou=Paris, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: Linda Charteris
description: Engineer
```

dn: cn=Marcello Conti, ou=Rome, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: Marcello Conti
description: Engineer

dn: cn=Pieter Jansen, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: Pieter Jansen
description: Engineer

dn: cn=Johan Klaassens, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: Johan Klaassens
description: Engineer

And it works!

The distinguishedName, or "dn", consists of the set of attributes that uniquely identify an entry. This set of attributes corresponds to the "path" that has to be traveled to reach that entry. The last result shown above, "Johan Klaassens", is found by starting to search at "o=MegaFix and c=NL", going to the department of "Engineering" and finally going to the division "Amsterdam".

Adding data to the hierarchy

MegaFix's sales in Amsterdam have increased and thus the need for a third engineer arose. Luckily, Mr. Wim Poorten applied for the job and got it.

To add Mr. Poorten to our directory, we create the file /etc/ldap/MegaFix.Amsterdam.Add.ldif, I like descriptive names, and issue the command:

ldapadd -f /etc/ldap/MegaFix.Amsterdam.Add.ldif -D 'cn=Manager,o=MegaFix,x=NL' -W -x
Enter LDAP Password: (which is 'blabla')

There should now be three engineers working in the Amsterdam division:

ldapsearch -LLL -b 'ou=amsterdam,ou=engineering,o=MegaFix,c=nl' -x cn
dn: ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL

dn: cn=Pieter Jansen, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: Pieter Jansen

dn: cn=Johan Klaassens, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: Johan Klaassens

dn: cn=Wim Poorten, ou=Amsterdam, ou=Engineering, o=MegaFix, c=NL
cn: Engineer
cn: Wim Poorten

And, as you can see, there are!

Changing data in the hierarchy

Unfortunately, sales are down in London but are expected to increase the next quarter. In Paris, there are a lot of problems, so an extra engineer for a week or two would be nice. John Hughes always wanted to see Paris and offered to assist his colleague Linda for a week or two.

Naturally, the directory should reflect this new situation to enable people to locate John when needed. Since the "ou" is part of the distinguishedName and ou occurs twice, a simple modify won't work. The safest way to do this is to get the existing entry and write it to a file in ldif format. Then, modify that file to first delete the existing entry and then add a new one with the new location.

```
# ldapsearch -LLL -b 'ou=london,ou=engineering,o=MegaFix,c=nl' -x > MegaFix.John.to.Paris.ldif
```

This gives all information about the London division of the Engineering department:

```
dn: ou=London, ou=Engineering, o=MegaFix, c=NL
objectClass: organization
description: Division London

dn: cn=John Hughes, ou=London, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: John Hughes
sn: Hughes
description: Engineer

dn: cn=Peter Baines, ou=London, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: Peter Baines
sn: Baines
description: Engineer
```

Remove the all entries except the one from John Huges and modify/add lines so that the file looks like this:

```
# Remove John Hughes from London
dn: cn=John Hughes, ou=London, ou=Engineering, o=MegaFix, c=NL
changetype: delete

# And place him in the Paris division
dn: cn=John Hughes, ou=Paris, ou=Engineering, o=MegaFix, c=NL
changetype: add
objectClass: person
cn: Engineer
cn: John Hughes
sn: Hughes
description: Engineer
```

To commit the changes to the database, we issue the command:

```
# ldapmodify -r -f MegaFix.John.to.Paris.ldif -D 'cn=Manager,o=MegaFix,c=NL' -W -x
Enter LDAP Password: (which is 'blabla')
deleting entry "cn=John Hughes, ou=London, ou=Engineering, o=MegaFix, c=NL"
adding new entry "cn=John Hughes, ou=Paris, ou=Engineering, o=MegaFix, c=NL"
```

To demonstrate that John is now stationed in Paris, we issue the following search command:

```
pug:/etc/ldap# ldapsearch -LLL -b 'ou=Paris,ou=engineering,o=MegaFix,c=nl' -x
dn: ou=Paris, ou=Engineering, o=MegaFix, c=NL
objectClass: organization
description: Division Paris

dn: cn=Linda Charteris, ou=Paris, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: Linda Charteris
sn: Charteris
description: Engineer

dn: cn=John Hughes, ou=Paris, ou=Engineering, o=MegaFix, c=NL
objectClass: person
cn: Engineer
cn: John Hughes
sn: Hughes
description: Engineer
```

More on LDAP

If you would like to read more about LDAP, this section points you to a few sources of information:

- *The OpenLDAP site*
- *The OpenLDAP Administrator's guide*
- *The LDAP Linux HOWTO*
- *The Internet FAQ archives* where RFC's and other documentation can be searched and found.

PAM authentication (2.210.4)

PAM authentication (2.210.4)

The candidate should be able to configure PAM to support authentication via traditional /etc/passwd, shadow passwords, NIS or LDAP.

Key files, terms and utilities include:

/etc/pam.d
/etc/pam.conf

What is it?

PAM stands for Pluggable Authentication Modules. PAM consists of a set of libraries and an API (Application programming Interface) that can be used to perform authentication tasks. Privilege granting programs, such as **login** and **su**, use the API to perform standard authentication tasks.

How does it work?

The authentication tasks can be subdivided into four different functional groups:

account

Provide account verification types of service: has the user's password expired?; Is this user permitted access to the requested service?

authentication

Establish if the user really is whom he claims to be. This can be done, for example, by asking a password or, given the right module, by reading a chip-card or by performing a retinal or fingerprint scan.

password

This group's responsibility is the task of updating authentication mechanisms. Typically, such services are strongly coupled to those of the authentication group. Some authentication mechanisms lend themselves well to being updated. The user might be presented with a question like "Please enter the new password".

session

This group of tasks covers things that should be done prior to a service being offered and after it is withdrawn. Such tasks include the maintenance of audit trails and the mounting of the user's home directory. The session management group is important as it provides both an opening and closing hook for modules that affect the services available to a user.

PAM can be configured using the file /etc/pam.conf which has the following format:

service type control module-path module-arguments

The meaning of these five fields is:

service

This is the name of the application involved, for example: **login**, **ssh** or **passwd**.

type

This is the type of group the task to be performed belongs to: account, auth (the authentication group), password or session.

control

This field indicates what the PAM-API should do in case authentication fails for any module.

The are four valid values for the control field:

requisite

Upon failure, the authentication process will be terminated immediately.

required

This will return failure after the remaining modules for this service and type have been invoked.

sufficient

Upon success, the authentication process will be satisfied, even if a prior required module has failed the authentication.

optional

The success or failure of this module is only important if this is the only module associated with this service and this type.

module-path

This is the filename, including the full path, of the PAM that is to be used by the application.

module-arguments

These are module specific arguments, separated by spaces, that are to be passed to the module. Refer to the specific module's documentation for further details.

Configuration is also possible using individual configuration files, which is recommended. These files should all be located in the /etc/pam.d directory. If this directory exists, the file /etc/pam.conf will be ignored. The filenames should all be lowercase and be identical to the name of the service, such as login. The format of these files is identical to /etc/pam.conf with the exception that there is no service field.

Configuring authentication via /etc/passwd and /etc/shadow

What we need is the unix authentication module pam_unix.so which uses standard calls from the system's libraries to retrieve/set account information/authentication. This information is obtained from the files /etc/passwd and, if shadow passwords are enabled, /etc/shadow.

The pam_unix.so module supports the following management groups:

account

The type "account" does not authenticate the user but checks other things such as the expiration date of the password and might force the user to change his password based on the contents of the files /etc/passwd and /etc/shadow.

The following options are supported:

debug

Log information using **syslog**.

audit

Also logs information, even more than debug does.

auth

The type "auth" checks the user's password against the password database(s). This component is configured in the file /etc/nsswitch.conf. Please consult the man page (**man nsswitch.conf**) for further details.

The following options are supported:

audit

Log information using **syslog**.

debug

Also logs information using **syslog** but less than audit.

nodelay

This argument sets the delay-on-failure, which has a default of a second, to nodelay.

nullok

Allows empty passwords. Normally authentication fails if the password is blank.

try_first_pass

Use the password from the previous stacked auth module and prompt for a new password if the retrieved password is blank or incorrect.

use_first_pass

Use the result from the previous stacked auth module, never prompt the user for a password and fails if the result was a fail.

password

The type "password" changes the user's password.

The following options are supported:

audit

Log information using **syslog**.

bigcrypt

Use the DEC "C2" extension to crypt().

debug

Also logs information using **syslog** but less than audit.

md5

Use md5 encryption instead of crypt().

nis

Use NIS (Network Information Service) passwords.

not_set_pass

Don't use the passwords from other stacked modules and do not give the new password to other stacked modules.

nullok

Allows empty passwords. Normally authentication fails if the password is blank.

remember

Remember the last n passwords to prevent the user from using one of the last n passwords again.

try_first_pass

Use the password from the previous stacked auth module, and prompt for a new password if the retrieved password is blank or incorrect.

use_authtok

Set the new password to the one provided by a previous module.

use_first_pass

Use the result from the previous stacked auth module, never prompt the user for a password and fails if the result was a fail.

session

The type "session" uses syslog to log the user's name and session type at the start and end of a session.

The "session" type does not support any options.

For each service that requires authentication a file with the name of that service must be created in /etc/pam.d. Examples of those services are: **login**, **ssh**, **ppp**, **su**.

For example purposes the file /etc/pam.d/login will be used:

```
# Perform password authentication and allow accounts without a password
auth      required   pam_unix.so nullok

# Check password validity and continue processing other PAM's even if
# this test fails. Access will only be granted if a 'sufficient' PAM,
# that follows this 'required' one, succeeds.
account   required   pam_unix.so

# Log the user name and session type to syslog at both the start and the end
# of the session.
session   required   pam_unix.so
```

```
# Allow the user to change empty passwords (nullok), perform some additional
# checks (obscure) before a password change is accepted and enforce that a
# password has a minimum (min=4) length of 4 and a maximum (max=8) length of
# 8 characters.
password   required   pam_unix.so nullok obscure min=4 max=8
```

Configuring authentication via NIS

To be able to authenticate via NIS, the module pam_nis.so is needed. This module can be found at *PAM NIS Authorisation Module*.

To set up things in such a way that NIS authentication is sufficient (and if that is not the case try pam_unix.so), the lines that do the trick in /etc/pam.d/login are:

```
auth    sufficient pam_nis.so item=user \
              sense=allow map=users.byname value=compsci
auth    required   pam_unix.so try_first_pass

account sufficient pam_ldap.so \
              item=user sense=deny map=cancelled.byname error=expired
account required   pam_unix.so
```

Configuring authentication via LDAP

To be able to authenticatie via LDAP, the module pam_ldap.so is needed. This module can be found at *PADL Software Pty Ltd*.

To set up things in such a way that LDAP authentication is sufficient, (and if that is not the case try pam_unix.so), the lines that do the trick in /etc/pam.d/login are:

```
auth    sufficient pam_ldap.so
auth    required   pam_unix.so try_first_pass

account sufficient pam_ldap.so
account required   pam_unix.so
```

LDAP configuration (2.210.3)

Home

Chapter 11. System Maintenance (2.211)

Chapter 11. System Maintenance (2.211)

Prev Next

Chapter 11. System Maintenance (2.211)

Revision: $Revision: 1.8 $ ($Date: 2004/03/03 15:03:36 $)

This topic has a total weight of 4 points and contains the following objectives:

Objective 2.211.1; System logging (1 point)

> The candidate should be able to configure syslogd to act as a central network log server. This objective also includes configuring syslogd to send log output to a central log server, logging remote connections and using grep and other text utilities to automate log analysis.

Objective 2.211.2; Packaging software (1 point)

> The candidate should be able to build a package. This objective includes building (or rebuilding) both RPM and DEB packaged software.

Objective 2.211.3; Backup Operations (2 points)

> The candidate should be able to create an off-site backup storage plan.

System Logging (2.211.1)

The candidate should be able to configure syslogd to act as a central network log server. This objective also includes configuring syslogd to send log output to a central log server, logging remote connections and using grep and other text utilities to automate log analysis.

Key files, terms and utilities include:

syslog.conf
sysklogd
/etc/hosts

Resources: the man pages of **syslogd(1)**, **klogd(1)** and **syslog.conf(5)**.

Sysklogd

Sysklogd is an enhanced version of the standard Berkeley utility program which is used on Linux systems. It is responsible for logging messages received from programs and facilities on the local host, as well as from remote hosts.

Sysklogd includes two system utilities **syslogd** and **klogd**, which support system logging and kernel message trapping. Support of both internet and unix domain sockets enables this utility package to support both local and remote logging.

syslogd

syslogd provides the kind of logging that is used by many modern programs. Every logged message contains at least a time, a hostname and, usually, a program name field. (See the syslog(3) manual page for information on generating messages)

Rules in /etc/syslog.conf specify where messages should go. An example syslog.conf entry is shown below:

```
# Sample syslog.conf line
daemon.warning      /var/log/daemon.log
```

This means that messages of priority *warning* and higher from the facility *daemon* will be sent to the file /var/log/daemon.log.

A message is sent to the destinations of all matching rules.

The facility specifies the subsystem that produces the message. It is one of the following keywords:

auth

Security and authorization messages. This facility is deprecated, use *authpriv* instead.

authpriv

Security and authorization messages.

cron

Clock deamons (**at** and **cron**).

daemon

Miscellaneous deamons without a separate facility value.

ftp

Ftp deamons.

kern

Kernel messages.

lpr

Line printer subsystem.

mail

Mail subsystem, including services like **fetchmail** and IMAP.

news

USENET news subsystem.

syslog

Messages generated internally by **syslogd**.

user

Generic user-level messages. This is the default.

uucp

UUCP subsystem.

local0 through *local7*

Reserved for local use.

The priority is one of the following keywords, in ascending order: *debug*, *info*, *notice*, *warning*, *err*, *crit*, *alert*, *emerg*. The priority defines the severity of the message. The standard behavior is that all messages of the specified priority and higher are logged to the given action.

The second field is the "logfile", but it need not be a regular file. The **syslogd** provides the following actions:

Regular File

Typically messages are logged to real files. The file has to be specified with the full pathname, beginning with a slash ('/').

You may prefix an entry with the minus ('-') sign to omit syncing the file after every logging. Note that you might lose information if the system crashes right after a write attempt. Nevertheless, this might give you back some performance, especially if you run programs that use logging in a very verbose manner.

Terminal and Console

If the file you specified is a tty, special tty-handling is done, same as with /dev/console.

Remote Machine

syslogd provides full remote logging, that is, it is able to send messages to a remote host running **syslogd** and to receive messages from remote hosts. The remote host won't forward the message again, it will just log them locally. To forward messages to another host, prepend the hostname with the at sign ('@'):

```
# Send all logging to a remote machine
*.*   @remote
```

Using this feature, you're able to control all syslog messages on one host, assuming all other machines log remotely to that host. This makes administration easier.

If the remote hostname cannot be resolved at startup because the name-server is inaccessible (perhaps it is started after **syslogd**), no need to worry: **syslogd** will retry ten times before complaining. One way to avoid this is to place the hostname in /etc/hosts.

Note that you have to start the central logging server with the -r option to enable this. By default, **syslogd** doesn't listen to the network. Also note that **syslogd** has no facility to split log messages from different hosts to different files.

List Of Users

Usually, critical messages are also directed to "root" on the host machine. You can specify a list of users who get the message by simply writing their login names. You may specify more than one user by separating them with commas (',').

```
# Messages of the priority alert will be directed to root and joey
*.alert     root,joey
```

If they're logged in, they get the message. An e-mail will not be sent, because, by the time it arrives, it might be too late.

Emergency messages often should go to all users currently online to notify them that something strange is happening with the system. To specify this **wall**-feature, use an asterisk ('*').

klogd

klogd is a system daemon which intercepts and logs linux kernel messages.

The kernel log information is gathered from either the /proc filesystem or from the syscall interface.

Although **klogd** has the option of sending the log messages directly to a file, the default behavior is to send the messages to **syslogd** using the *kern* facility.

Prev | Next
PAM authentication (2.210.4) | Home | Packaging Software (2.211.2)

Packaging Software (2.211.2)

The candidate should be able to build a package. This objective includes building (or rebuilding) both RPM and DEB packaged software.

Key files, terms and utilities include:

rpm
SPEC file format
/debian/rules

Resources: Jackson01

While there is a vast quantity of software available in the DEB or RPM format, sometimes it is necessary to install software that isn't. It is often possible to install the software from source, but you may want to create a package from the source and install the package. This is useful if you want to be able to do updates without worrying if all obsolete files are removed or if the software needs to be installed on more than one machine.

Both DEB and RPM use a similar structure to build a package. A script, in one way or another, is used to build the package to run in /usr and install it in a temporary directory (DEB: <sourcedir>/debian/<packagename> ; RPM: /var/tmp/<packagename>-buildroot). The appropriate files are gathered in an archive and some control data is added. The resulting DEB or RPM can be installed on the target platform.

Both package systems start from the original source and apply a patch where necessary to correct any problems that might arise in packaging.

The major difference between the packaging formats is the location and structure of the "scripts". In the DEB format, a directory (debian) which contains a number of files is added to the source. RPM uses a spec-file, which is separate from the source.

DEB Packages

As mentioned in the "Debian Policy Guide" (Jackson01), the debian directory structure can be generated with **dh_make**. This utility will try to guess a suitable debian/rules file based on information in the source-directory.

The generated debian directory contains a number of files. Some files will need to be edited, e.g., the *Description:* needs a proper description in the control file and documentation you want to have installed in /usr/share/doc/*packagename* will have to be added to the docs file. See Jackson01 for more information.

Only the required files will be mentioned here. These are control, changelog, copyright and rules.

control file

This file contains various values which **dpkg** and **dselect** will use to manage the package. Here is an example:

```
Source: bao
Section: games
Priority: optional
Maintainer: Ben Mesman <ben@mesman.nu>
Build-Depends: debhelper (>> 3.0.0)
Standards-Version: 3.5.2

Package: bao
Architecture: any
Depends: ${shlibs:Depends}
Description: A mancala game from Tanzania and Zanzibar
 Bao is what we call a mancala game. Mancala is the term to denominate
 games with one shared characteristic: moves are not executed as in
 chess or checkers, instead moves are executed by sowing seeds (or
 other playing pieces) into holes.
```

The first six lines are the control information for the source package.

The *Section:* field is the Debian section this package goes to. This package will be installed in the section "games". Other sections include for instance "x11" for X11 specific programs or "devel" for programmer tools.

The *Priority:* field describes how important it is for the user to install this package. Possible priorities are "required", "important", "standard", "optional" and "extra". This should probably be "optional" or "extra" for self-created packages.

The *Maintainer:* field is the maintainer of the Debian package, not the upstream maintainer. The *Standards-Version* is the version of the Debian Policy Manual the package conforms to.

If you want to resolve the build dependencies (i.e., the packages needed to build this

package) automatically, you need to specify the packages in the *Build-Depends* field.

The second part of the file contains information on the binary package that will be generated. If more binary packages should be created from the same source, there will be more of these sections.

Note that the *Architecture:* field should be set to "all" for most packages. Se the Debian Policy Manual for more information on architecture dependent packages.

Apart from the package description, the most important information here is the relationship with other packages. There are seven possible relations:

Depends:

Used if the program absolutely will not run (or will cause severe breakage) unless a particular package is present.

Recommends:

Used for packages that are not strictly necessary but are typically used with the program.

Suggests:

Used for packages which will work nicely with the program but are not at all necessary

Pre-Depends:

This is stronger than Depends:. The package will not be installed unless the packages it pre-depends on are installed *and correctly configured*. Use this very sparingly and only after discussing it on the debian-devel mailing list.

Conflicts:

Used if the program absolutely will not run (or will cause severe breakage) if a particular package is present.

Provides:

For some types of packages where there are multiple alternatives, virtual names have been defined. You can get the full list in /usr/share/doc/debian-policy/virtual-package-names-list.text.gz file. Used if the program provides a function of an existing virtual package.

Replaces:

Used when the program replaces files from another package or completely replaces another package (used in conjunction with Conflicts:). Files from the named packages will be removed before installing.

Note that in this example the "Depends:" field contains $(shlibs:Depends). This will be

automatically generated by **dh_shlibdeps**, and filled in by **dh_gencontrol**, with the names of any shared libraries the program uses, such as libc6 or xlib6g, so they don't have to be explicitly specified.

copyright file

This file contains the information about package upstream resources, copyright and license information. Its format is not dictated by the Debian Policy[Jackson01], but the contents is (section 6.5). **dh_make** creates a default one. This is what it looks like:

This package was debianized by Ben Mesman <ben@snow.nl>
on Fri, 23 Nov 2001 11:42:37 +0100.

It was downloaded from <fill in ftp site>

Upstream Author(s): <put author(s) name and email here>

Copyright:

<Must follow here>

If the program is licensed under one of the common free software licenses, such as the GNU GPL or LGPL, BSD or the Artistic license, you can refer to the appropriate file in the /usr/share/common-licenses/ directory. If the program has its own license, the complete license must be included in this file.

Changelog file

This is a required file and uses a special format. This format is used by dpkg and other programs to obtain the version number, revision, distribution and urgency of the package.

For the maintainer of the program, it is also important, since it is good to document all changes. It will help people downloading the package to see whether there are any unresolved issues that they should know about before installing. It will be saved as /usr/share/doc/bao/changelog.Debian.gz in the binary package.

This is an example Changelog:

bao (0.1.0) unstable; urgency=low

  * Initial Release.

 -- Ben Mesman <ben@snow.nl> Fri, 23 Nov 2001 11:42:37 +0100

Local variables:
mode: debian-changelog
End:

The first line is the package name, version, distribution and urgency. The name must match the source package name, distribution should be either "unstable" or "experimental" (for now), and urgency probably shouldn't be changed to anything higher than "low".

The third line is a log entry describing the changes in this release. The fifth line closes the comments and specifies who made the release and when. The *name <email>* must be followed by two spaces and the date in RFC822 format (e.g., as generated with **date -R**).

New releases should be added at the top.

rules file

Now we need to take a look at the exact rules which **dpkg-buildpackage** will use to actually create the package. This file is actually another Makefile, since it is executed with "make -f", but different from the one in the upstream source.

The important part to know about the rules file created by **dh_make** is that it is just a suggestion. It will work for simple packages, but, for more complicated ones, don't be afraid to add and subtract from it to fit your needs. The only thing that you must not change are the names of the rules, because all the tools use these names, as mandated by the Packaging Manual.

```
#!/usr/bin/make -f
# Sample debian/rules that uses debhelper.
# GNU copyright 1997 to 1999 by Joey Hess.

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

# This is the debhelper compatibility version to use.
export DH_COMPAT=3

configure: configure-stamp
configure-stamp:
	dh_testdir
	# Add here commands to configure the package.
	./configure --prefix=/usr --mandir=\$${prefix}/share/man --infodir=\$${prefix}/share/info

	touch configure-stamp
```

```
build: configure-stamp build-stamp
build-stamp:
	dh_testdir

	# Add here commands to compile the package.
	$(MAKE)
	#/usr/bin/docbook-to-man debian/bao.sgml > bao.1

	touch build-stamp

clean:
	dh_testdir
	# dh_testroot
	rm -f build-stamp configure-stamp

	# Add here commands to clean up after the build process.
	-$(MAKE) distclean

	dh_clean

install: build
	dh_testdir
	dh_testroot
	dh_clean -k
	dh_installdirs

	# Add here commands to install the package into debian/bao.
	$(MAKE) install DESTDIR=$(CURDIR)/debian/bao

# Build architecture-independent files here.
binary-indep: build install
# We have nothing to do by default.

# Build architecture-dependent files here.
binary-arch: build install
	dh_testdir
	dh_testroot
#	dh_installdebconf
	dh_installdocs
	dh_installexamples
	dh_installmenu
#	dh_installlogrotate
```

```
#      dh_installemacsen
#      dh_installpam
#      dh_installmime
#      dh_installinit
       dh_installcron
       dh_installman
       dh_installinfo
#      dh_undocumented
       dh_installchangelogs
       dh_link
       dh_strip
       dh_compress
       dh_fixperms
#      dh_makeshlibs
       dh_installdeb
#      dh_perl
       dh_shlibdeps
       dh_gencontrol
       dh_md5sums
       dh_builddeb

binary: binary-indep binary-arch
.PHONY: build clean binary-indep binary-arch binary install configure
```

As you can see from the first line, this script will be processed by **make**.

The "configure" rule (and its child, "config-stamp") specify that the program will be configured to run in /usr (instead of the default /usr/local). Also the manual pages and the shared files will be installed in the Debian default location.

The "build" rule (and its child "build-stamp") specify how the program is to be built. In most cases, the program's own Makefile will be used.

The "clean" rule cleans up any unneeded binary or auto-generated stuff left over from building the package. This rule must be working at all times (even when the source tree is cleaned up!), so please use the forcing options (e.g., for **rm**, that is *-f*), or ignore return values (with a - in front of a command name).

The installation process, the "install" rule, basically runs the "install" rule from the program's own Makefile, but installs in <pwd>/debian/bao directory.

As the comments suggest, the "binary-indep" rule is used to build architecture independent packages. As none are specified in the control file, nothing will be done. If the package was an "Architecture: all" one, all the commands for building the package would be under this

rule, and the next rule (“binary-arch”) would be empty instead.

The “binary-arch” rule uses several small utilities from the *debhelper* package to perform various operations to make the package Policy conforming.

The names start with dh_, and the rest is the description of what the particular utility really does. It is all quite self-explanatory, but here are some additional explanations:

- dh_testdir checks that you are in the right directory (i.e. the top-level source directory),
- dh_testroot checks that you have root permissions which is needed for *binary** targets, and *clean*,
- dh_installmanpages copies all the manpages it can find in the source tree to package,
- dh_strip strips debugging headers from executable files and libraries, to make them smaller,
- dh_compress gzips manual pages and documentation larger than 4 Kb,
- dh_installdeb copies package related files (e.g. the maintainer scripts) under debian/tmp/DEBIAN directory,
- dh_shlibdeps calculates shared libraries dependencies of the libraries and executables,
- dh_gencontrol adds stuff to, and installs, the control file,
- dh_md5sums generates MD5 checksums for all the files in the package.

For more information on these and other debhelper commands, read the debhelper documentation.

Building The Package

When the rules file is adapted to the program, a package can be build by executing

dpkg-buildpackage -rfakeroot

This will do everything for you, including the signing of the files. You will just have to enter your PGP (or GPG) secret key twice.

In our example, the following files will be generated:

bao_0.1.0-1_i386.deb

: This is the completed binary package. You can use **dpkg** or **apt** to install or remove this just like any other package;

bao_0.1.0.orig.tar.gz

: This is the original source code gathered up so that if someone else wants to recreate the package from scratch they can. Or if they aren't using the Debian packaging

system, but need to manually download the source and compile.

bao_0.1.0-1.dsc

This is a summary of the contents of the source code. The file is generated from the bao-0.1.0/debian/control file and is used when unpacking the source with **dpkg-source**. This file is PGP signed, so that people can be sure of its origin.

bao_0.1.0-1.diff.gz

This compressed file contains each and every addition made to the original source code, in the form known as "unified diff". It is made and used by **dpkg-source**.

bao_0.1.0-1_i386.changes

This file describes all the changes made in the current package revision, and it is used by the Debian FTP archive maintenance programs to install the binary and source packages in it. It is partly generated from the bao-0.1.0/debian/changelog file and the .dsc file.

People downloading the package can look at this file and quickly see what has changed. The long strings of numbers are MD5 checksums for the files mentioned. Those downloading the files can test them with **md5sum** and if the numbers don't match, they'll know the file is corrupt or has been hacked. This file is PGP signed, so that people can be even more sure that it is authentic.

RPM Packages

The SPEC file is a description of how a (source-)RPM can be built from source. The SPEC file should be named according to a standard convention.

<program name>-*<version number>*-*<release number>*.spec

This will ensure that if you install multiple source RPMs for different versions of the same package that the spec files remain intact.

Before you can actually build an RPM from a spec file, you should make sure you have a proper build tree. It should contain the following directories:

- BUILD is the directory where all building occurs by RPM. You don't have to do your test building anywhere in particular, but this is where RPM will do its building,
- SOURCES is the directory where you should put your original source tar files and your patches. This is where RPM will look by default,
- SPECS is the directory where all spec files should go,
- RPMS is where RPM will put all binary RPMs when built,
- SRPMS is where all source RPMs will be put.

The default location is of the build tree is /usr/src.

Here is a sample of a spec file (bao-0.1.0-1.spec):

```
Summary: A mancala game from Tanzania and Zanzibar
Name: bao
Version: 0.1
Release: 1
Copyright: GPL
Group: Amusements/Games
Source: ftp://ftp.somesite.com/pub/games/bao-0.1.0.tar.gz
# Patch: none
BuildRoot: /var/tmp/%{name}-buildroot

%description
Bao is what we call a mancala game. Mancala is the term to denominate
games with one shared characteristic: moves are not executed as in
chess or checkers, instead moves are executed by sowing seeds (or
other other playing pieces) into holes.

%prep
%setup

%build
./configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info
make

%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$(RPM_BUILD_ROOT)

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root)
%doc README TODO

/usr/bin/bao

%changelog
* Tue Dec 04 2001 Ben Mesman <ben@snow.nl>
- initial release
```

The Header

The header consists of a series of fields that provide general information about the package. These fields are:

- *Summary:* This is a one line description of the package,
- *Name:* This is the <name> string from the rpm file,
- *Version:* This is the <version> string from the rpm file,
- *Release:* This is the <release> string from the rpm file,
- *Copyright:* This is the copyright of the package. It should be a short description, like GPL, BSD, MIT, public domain, distributable or commercial,
- *Group:* This is used by higher-level installation programs, such as **gnorpm**, to place this package in its proper place in the hierarchical structure. A description of the hierarchy can be found in /usr/doc/rpm*/GROUPS,
- *Source:* This is the location of the original source file. The filename on the local system must match the filename in this line (i.e., do not rename the file after downloading). Several source files can be specified along these lines:

```
Source0: http://site.name/downloads/hopla-01.tar.gz
Source1: http://site.name/downloads/hopla-02.tar.gz
Source2: ftp://other.site/pub/patches/addition-1.tar.gz
```

- *Patch:* This works basically, the same as the "Source:" field. It designates the location of the patch file. Multiple patches can be specified in the same manner as in "Source:",
- *BuildRoot:* This is the "root" for building and installing the new package. This can help test the package before it is installed,
- *%description* This is not really a header item, but should be described with the header anyway. It is a multi-line description of the package. There should be one "% description" field per package and/or subpackage.

Prep

This is the second section of the spec file. It should contain all commands to set-up the source to do a **make**. This is where you unpack the source(s) and apply the patch(es).

One thing to note is that each of these sections is really just a place to execute shell scripts. So, it is possible to create a sh script and put it after the **%prep** tag to unpack and patch the sources. However, there are macros to aid in this.

The first macro, shown in the example, is the **%setup** macro. It unpacks the sources mentioned in the "Source:" line in the header and **cd**'s into the source directory.

The second macro is called **%patch**. It will apply all patches mentioned in the header. Since there are no patches for this package, the **%patch** macro is not included in the example.

If something needs to be done that can't be handled by these macros, you can include another setup. Everything up to the "%build" section will be interpreted as an sh script.

Build

There aren't really any macros for this section. Just use any commands here that you would need to build the software once you have untarred the source, patched it and cd'ed into the directory. This is just another set of commands passed to sh, so any legal sh commands can go here (including comments).

> **Important**
>
> It is important to keep in mind that your current working directory is reset in each of these sections to the top level of the source directory. You can always **cd** into subdirectories if necessary.

Install

There aren't really any macros here, either. Basically, use any commands that are necessary for the install. If you have **make install** available to you in the package you are building, put that here. If not, you can either patch the Makefile for a **make install** and just do a **make install** here, or you can hand install them here with **sh** commands. You can consider your current directory to be the top-level of the source directory.

The variable RPM_BUILD_ROOT is available to tell you the path set as the "Buildroot:" in the header. Using build roots are optional, but are highly recommended because they keep you from cluttering your system with software that isn't in your RPM database (building an RPM doesn't touch your database...you must install the binary RPM you just built to do that).

Clean

It's a good idea to always make sure there is a clean build root before building a package a second time on a system. The "%clean" macro will help with that. Simply put the proper commands there to blow away a former build root.

In the example, the RPM_BUILD_ROOT is removed. To be prudent, it is a good idea to check that RPM_BUILD_ROOT was not set to / before doing something this volatile.

Files

This is the section where you *must* list the files for the binary package. RPM has no way of knowing what binaries get installed as a result of make install. To circumvent this, some have suggested doing a find before and after the package install. With a multiuser system, this is unacceptable as other files may be created during a package building process that

have nothing to do with the package itself.

There are some macros available to do some special things as well. They are listed and described here:

- **%doc** is used to mark documentation in the source package that you want installed in a binary install. The documents will be installed in /usr/doc/$NAME-$VERSION-$RELEASE. You can list multiple documents on the command line with this macro, or you can list them all separately using a macro for each of them.
- **%config** is used to mark configuration files in a package. This includes files like sendmail.cf, passwd, etc. If you later uninstall a package containing config files, any unchanged files will be removed and any changed files will get moved to their old name with a .rpmsave appended to the filename. You can list multiple files with this macro as well.
- **%dir** marks a single directory in a file list to be included as being owned by a package. By default, if you list a directory name *without* a **%dir** macro, *everything* in that directory is included in the file list and later installed as part of that package.
- **%defattr** allows you to set default attributes for files listed after the **%defattr** declaration. The attributes are listed in the form (mode, owner, group) where the mode is the octal number representing the bit pattern for the new permissions (such as **chmod** would use), owner is the username of the owner, and group is the group you would like assigned. (You may leave any field to the installed default) by simply placing a "-" in its place, as was done in the mode field for the example package.
- **%files -f <filename>** will allow you to list your files in some arbitrary file within the build directory of the sources. This is nice in cases where you have a package that can build its own file list. You then include that file list here and you won't have to specifically list the files.

A potential problem in the file list is listing directories. If you list /usr/bin by accident, your binary package will contain every file in /usr/bin on your system.

Changelog

This is a log of the changes that were made when the package was updated. If you are modifying an existing RPM it is a good idea to list the changes here.

The format is simple. Start each new entry with a line with a "*" followed by the date, your name and your email address. The date should appear in the same format that is output by:

date +"%a %b %d %Y"

The rest of the section is a free text field, but should be organized in some coherent manner.

Building RPMs

When the SPEC file is finished, you can build an RPM with:

rpm -bb <spec-file>

The -bb indicate that **rpm** should *b*uild a *b*inary package. -bs builds a source rpm and -ba builds both (all). The RPM will be placed in /usr/src/RPMS/ and the source RPM will be placed in /usr/src/SRPMS (or wherever your RPM-build tree should be).

Prev | Up | Next

Chapter 11. System Maintenance (2.211) | Home | Backup Operations (2.211.3)

Backup Operations (2.211.3)

The candidate should be able to create an off-site backup storage plan.

References: LinuxRef07, Wirzenius98.

Why?

Everyone (more or less) knows that, as a system administrator, it is vital to make backups. Most also know why. Your data is valuable. It will cost you time and effort to re-create it, and that costs money or at least personal grief and tears. Sometimes it can't even be re-created, such as the results of some experiments. Since it is an investment, you should protect it and take steps to avoid losing it.

There are four main reasons why you may lose data: human error, software bugs, hardware failure and natural disasters. Humans are quite unreliable, they might make a mistake or be malicious and destroy data on purpose. Modern software does not even pretend to be reliable. A rock-solid program is an exception, not a rule. Hardware is more reliable, but may break seemingly spontaneously, often at the worst possible time. Nature may not be evil, but, nevertheless, can be very destructive sometimes.

What?

In general, you want to back up as much as possible. A major exception is the /proc filesystem. Since this only contains data that the kernel generates automagically, it is never a good idea to back it up. The /proc/kcore file is especially unnecessary, since it is just an image of your current physical memory; it's pretty large as well. Some special files that are constantly changed by the operating system (e.g. /etc/mtab) should not be restored, hence not be backed up. There may be others on your system.

Gray areas include the news spool, log files and many other things in /var. You must decide what you consider important. Also, consider what to do with the device files in /dev. Most backup solutions can backup and restore these special files, but you may want to re-generate them with a script.

The obvious things to back up are user files (/home) and system configuration files (/etc, but possibly other things scattered all over the filesystem).

When?

Depending on the rate of change of the data, this may be anything from daily to almost never. The latter happens on firewall systems, where only the log files change daily (but logging should happen on a different system anyway). So, the only time when a backup is necessary, for example, is when the system is updated or after a security update.

On normal systems, a daily backup is often best.

Backups can take several hours to complete, but, for a successful backup strategy, human interaction should be minimal, preferably just a matter of placing a tape in the tape device.

How?

While the brand or the technology of the hardware and software used for backups is not important, there are, nevertheless, important considerations in selecting them. Imagine, for example, the restore software breaks and the publisher has since gone out of business.

No matter how you create your backups, the two most important parts in a backup strategy are:

verifying the backup

> The safest method is to read back the entire backup and compare this with the original files. This is very time-consuming and often not an option. A faster, and relatively safe method, is to create a table of contents (which should contain a checksum per file) during the backup. Afterwards, read the contents of the tape and compare the two.

testing the restore procedure

> This means that you *must* have a restore procedure. This restore procedure has to specify how to restore anything from a single file to the whole system. Every few months, you should test this procedure by doing a restore.

Where?

If something fails during a backup, the medium will not contain anything useful. If this was your only medium, you are screwed. So you should have at least two sets of backup media. But if you store both sets in the same building, the first disaster that comes along will destroy all your precious backups along with the running system.

So you should have at least one set stored at a remote site. Depending on the nature of your data, you could store weekly or daily sets remotely.

Do not forget to store a copy of the backup-plan along with the backups at the remote site. Otherwise you cannot guarantee that the system will be back up-and-running in the shortest possible time.

Prev | Up | Next
Packaging Software (2.211.2) | Home | Chapter 12. System Security (2.212)

Chapter 12. System Security (2.212)

Prev Next

Chapter 12. System Security (2.212)

Revision: $Revision: 1.13 $ ($Date: 2004/08/13 08:15:14 $)

This topic has a total weight of 10 points and contains the following objectives:

(Objective 2.212.1 - not defined by LPI)

This objective was not defined by LPI

Objective 2.212.2; Configuring a router (2 points)

The candidate should be able to configure ipchains and iptables to perform IP masquerading and state the significance of Network Address Translation and Private Network Addresses in protecting a network. This objective includes configuring port redirection, listing filtering rules and writing rules that accept or block datagrams based upon source or destination protocol, port and address. Also included is saving and reloading filtering configurations, using settings in /proc/sys/net/ipv4 to respond to DOS attacks, using /proc/sys/net/ipv4/ip_forward to turn IP forwarding on and off and using tools such as PortSentry to block port scans and vulnerability probes.

Objective 2.212.3; Securing FTP servers (2 points)

The candidate should be able to configure an anonymous download FTP server. This objective includes configuring an FTP server to allow anonymous uploads, listing additional precautions to be taken if anonymous uploads are permitted, configuring guest users and groups with chroot jail and configuring ftpaccess to deny access to named users or groups.

Objective 2.212.4; Secure shell (OpenSSH) (2 points)

The candidate should be able to configure sshd to allow or deny root logins and enable or disable X forwarding. This objective includes generating server keys, generating a user's public/private key pair, adding a public key to a user's authorized_keys file and configuring ssh-agent for all users. Candidates should also be able to configure port forwarding to tunnel an application protocol over ssh, configure ssh to support the ssh protocol versions 1 and 2, disable non-root logins during system maintenance, configure trusted clients for ssh logins without a password and make multiple connections from multiple hosts to guard against loss of connection to remote host following configuration changes.

Objective 2.212.5; TCP_wrapper (1 point)

The candidate should be able to configure tcpwrappers to allow connections to specified servers from only certain hosts or subnets.

Objective 2.212.6; Security tasks (3 points)

The candidate should be able to install and configure kerberos and perform basic security auditing of source code. This objective includes arranging to receive security alerts from Bugtraq, CERT, CIAC and other sources, being able to test for open-mail relays and anonymous FTP servers and installing and configuring an intrusion detection system, such as snort or Tripwire. Candidates should also be able to update the IDS configuration as new vulnerabilities are discovered and apply security patches and bug-fixes.

Sources of information: RFC1918

Configuring a router (2.212.2)

The candidate should be able to configure ipchains and iptables to perform IP masquerading and state the significance of Network Address Translation and Private Network Addresses in protecting a network. This objective includes configuring port redirection, listing filtering rules and writing rules that accept or block datagrams based upon source or destination protocol, port and address. Also included is saving and reloading filtering configurations, using settings in /proc/sys/net/ipv4 to respond to DOS attacks, using /proc/sys/net/ipv4/ip_forward to turn IP forwarding on and off and using tools such as PortSentry to block port scans and vulnerability probes.

Key files, terms and utilities include:

ipchains
/proc/sys/net/ipv4
/etc/services
iptables
routed

Private Network Addresses

Why do Private Network Addresses exist ? It has been common practice to assign globally-unique addresses to all hosts that use TCP/IP. In order to extend the life of the IPv4 address space, address registries are requiring more justification concerning the need for extra address space than ever before, which makes it harder for organizations to acquire additional address space.

Hosts within enterprises that use IP can be partitioned into three categories:

Category 1

These hosts do not require access to the hosts of other enterprises or on the Internet itself; hosts within this category may use IP addresses that are unambiguous within an enterprise, but may be ambiguous between enterprises.

Category 2

These are hosts that need access to a limited set of outside services (e.g., E-mail, FTP, netnews, remote login), which can be handled by mediating gateways (e.g., application layer gateways). For many hosts in this category, unrestricted external access (provided via IP connectivity) may be unnecessary and even undesirable (for privacy/security reasons). These hosts, the same as category 1 hosts, may use IP addresses that are unambiguous within an enterprise, but may be ambiguous between enterprises.

Category 3

These hosts need network-layer access outside the enterprise (provided via IP connectivity); hosts in the last category require IP addresses that are globally unambiguous.

We will refer to the hosts in the first and second categories as "private" and to hosts in the third category as "public".

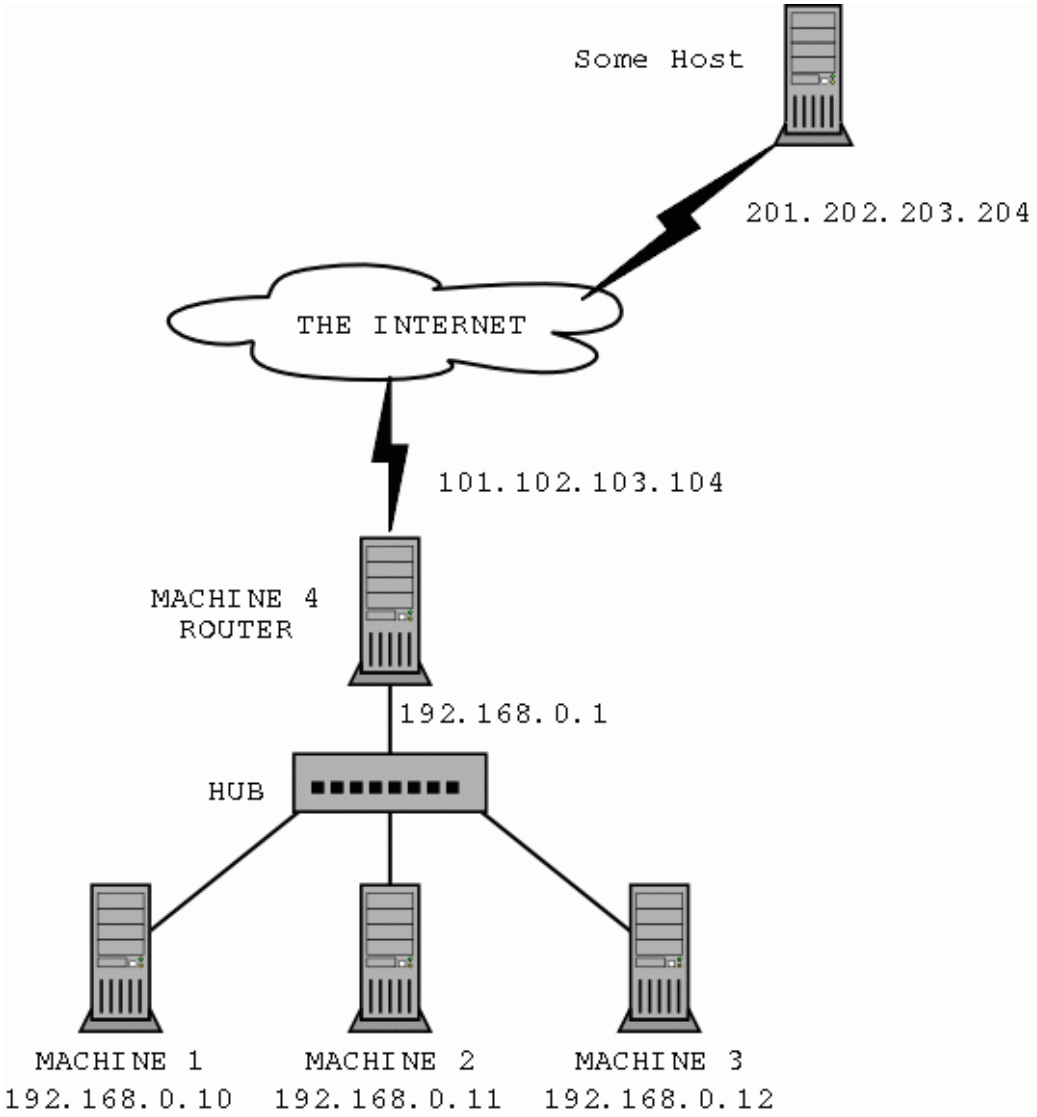
Many applications require connectivity only within one enterprise and do not need external (outside the enterprise) connectivity for the majority of internal hosts. In larger enterprises it is often easy to identify a substantial number of hosts using TCP/IP that do not need network-layer connectivity outside the enterprise.

The Internet Assigned Numbers Authority (IANA) has reserved the following three blocks of the IP address space for private internets:

10.0.0.0 - 10.255.255.255 (10/8 prefix)
172.16.0.0 - 172.31.255.255 (172.16/12 prefix)
192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

We will refer to the first block as "24-bit block", the second as "20-bit block", and to the third as "16-bit" block. Note that (in pre-CIDR notation) the first block is nothing but a single class A network number, while the second block is a set of 16 contiguous class B network numbers and third block is a set of 256 contiguous class C network numbers.

Network Address Translation (NAT)



The figure above displays s hypothetical situation which will serve as an example in the following explanation.

"The Firm" has four machines, 1 to 4, which are connected via a HUB and have private IP-Addresses in the 192.168.x.x range. Machine 4 serves as a router to the Internet and has two network interfaces. One connects the router to The Firm's internal network via the hub and has a private IP-address of 192.168.0.1, while the other connects the router to the Internet and has a valid (dynamic) IP-Address of 101.102.103.104.

Let's say that the user at Machine 2 wishes to look at a web page on Some Host (http://SomeHost.SomeCountry) with an IP-address of 201.202.203.204. To be able to see the web page, Machine 2 must be able to get information from the Internet and thus must be, in some way, connected to the Internet. And indeed, Machine 2 has an indirect connection to the Internet via Machine 4, but how can this work? Machine 2 has a private IP-address which is not supported on the Internet!

This is where NAT kicks in. Machine 4, the router, replaces the private IP-address of Machine 2 (and also of Machine 1 and 3 if needed) with its own IP-Address before sending the request to Some Host. Some Host thinks that a machine with IP-Address 101.102.103.104 asked for the web page and responds by sending the web page to Machine 4.

Machine 4 knows that it has replaced the IP-address of Machine 2 with its own before sending the request to Some Host so it also knows that the answer it got to the request has to go to Machine 2. Machine 4 accomplishes this by replacing its own IP-address in the answer by the IP-address of Machine 2.

This is, in a nutshell, what NAT does. For more detailed information consult RFC1631.

IP Masquerading with IPCHAINS

IP Masquerading is a form of NAT. To get IP Masquerading up and running on your Linux computer, your kernel must support this.

For more detailed information on compiling a kernel and modules, see Chapter 1, *Linux Kernel (2.201)* .

Let's go back to The Firm's network. The IP masquerading rules on machine 4 can be set using the following **ipchains** commands:

```
# ipchains -P forward DENY
# ipchains -A forward -i eth0 -s 192.168.0.0/24 -j MASQ
```

The first line defines a policy (-P) that states that all packets that have a destination address in the outside world will not be forwarded (DENY) by the forward chain (forward). This is done because it is good practice to forbid all traffic by default except traffic that you explicitly want.

The second line adds (-A) a rule to the forward chain (forward) that states that all packages arriving (-i) at interface eth0 and coming from (-s) an ip-address in the range 192.168.0.0-192.168.0.255 will be masqueraded (-j MASQ) and forwarded to the outside world. The "/24" states that the first 24 bits always have the same value, in this case 192.168.0, and that the last 8 bits can have any value.

Machine 4 now knows WHAT to do with IP packets that come from Machines 1 to 4, but not WHERE to send them. To complete Machine 4's instructions, we add the following commands:

```
# route add -net 192.168.0.0 netmask 255.255.255.0 eth0
# route add default gw 101.102.103.104 eth1
```

The first line adds a route to the 192.168.x.x network via eth0.

The second line adds a default route via eth1, which is used if the destination is not in the 192.168.x.x range.

IP forwarding with IPCHAINS

IP forwarding is the relaying of IP packets from one network to another. Let's go back to The Firm's network. We want Machine 4 to forward the IP packets to the Internet. This means that all of the Firms machines must use "real" IP addresses, as explained earlier, and that Machine 4 must know that certain IP addresses should go to the outside world, i.e., the Internet.

To enable forwarding, support must be enabled in the kernel (see Chapter 1, *Linux Kernel (2.201)* for more details) and forwarding itself must be enabled by issuing the command **echo 1 > /proc/sys/net/ipv4/ip_forward**.

Machine 4 now knows WHAT to do with IP packets that come from the Machines 1 to 4, but not WHERE to send them. To tell Machine 4 this, we use the following commands:

```
# route add -net 192.168.0.0 netmask 255.255.255.0 eth0
# route add default gw 101.102.103.104 eth1
```

The first line adds a route to the 192.168.x.x network via eth0. With kernel versions of 2.2 or higher this is done automatically.

The second line adds a default route via eth1 which is used if the destination is not in the 192.168.x.x range.

Port Redirection with IPCHAINS

Port redirection is a mechanism which enables the redirection of IP packets based on destination port number to another port. To enable port redirection, support must be enabled in the kernel. See the section called " IP Masquerading with IPCHAINS " for more details.

Since it worked so well the last time, let's go back to The Firm's network again for an example.

Suppose Machine 2 has a web-server (or some other program) running which listens to port 2345, and people from the outside must be able to connect to that program. Since The Firm is using private IP addresses for their internal network, Machine 2 is not visible on the Internet. The solution here is to tell Machine 4 to route all data from the outside that is aimed at port 80 to port 2345 on Machine 2.

Now we've got a problem: with the command **ipchains -j REDIR**, the port must be on the same host and here this is not the case. So, what to do now ?

The solution lies in the **ipmasqadm portfw** command. Typing **ipmasqadm portfw -h** gives the following information:

```
# ipmasqadm portfw -h
Usage: portfw -a -P PROTO -L LADDR LPORT -R RADDR RPORT [-p PREF] add entry
       portfw -d -P PROTO -L LADDR LPORT [-R RADDR RPORT]       delete entry
       portfw -f                                clear table
       portfw -l                                list table
       portfw <args> -n                             no names

PROTO is the protocol, can be "tcp" or "udp"
LADDR is the local interface receiving packets to be forwarded.
LPORT is the port being redirected.
RADDR is the remote address.
RPORT is the port being redirected to.
PREF  is the preference level (load balancing, default=10)
```

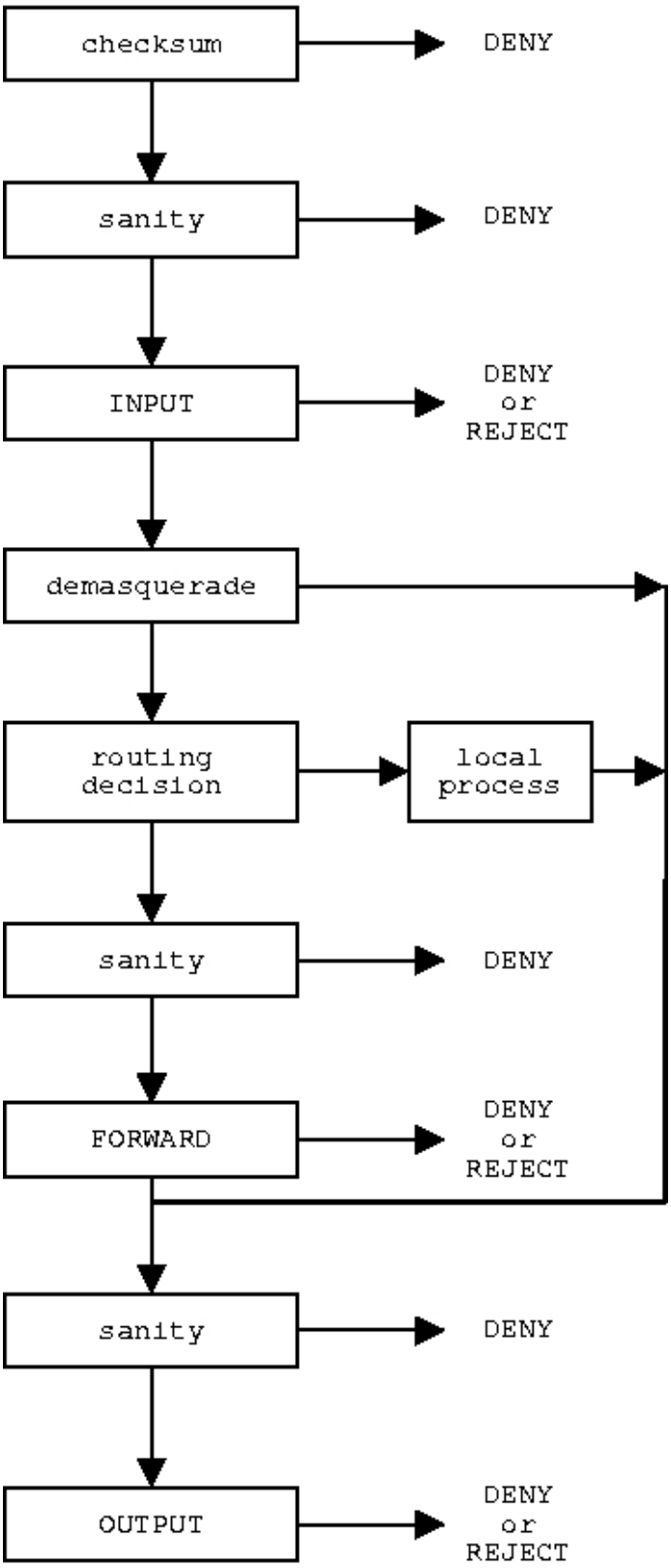
Executing the following commands on Machine 4 will achieve our goal:

```
# ipmasqadm portfw -f
# ipmasqadm portfw -a -P tcp -L 101.102.103.104 80 -R 192.168.0.11 2345
```

The first line empties the portfw table, the second line adds a portfw rule (portfw -a) for the tcp packets (-P tcp) arriving at port 80 on the external interface of Machine 4 (-L 101.102.103.104 80), which states that those packets must be sent to port 2345 on Machine 2 (-R 192.168.0.11 2345).

And what about the replies from Machine 2 in response to the requests? Shouldn't there be a rule for that too? The answer is no, because this works similarly to masquerading. Machine 2 has no idea whatsoever that the request came from the outside. To Machine 2, it looks like the request came from Machine 4, so the answer goes to Machine 4. Machine 4 knows that this is in answer to a request from the outside that has been rerouted to Machine 2, so it sends the answer to the same machine the request came from.

IPCHAINS, an overview



This figure shows the stages of the ipchains process.

IPCHAINS, nomen est omen, hence there are chains, a minimum of three to be precise. We've always got an INPUT, FORWARD and OUTPUT chain and, optionally, one or more user-defined chains.

A chain contains rules that determine the fate of a packet based on its source address, destination address, source port, destination port, protocol-type or any combination of these.

If a packet matches a rule, the packet is sent to the chain specified in the target. This can be ACCEPT, DENY, MASQ, REDIRECT or RETURN or a user-defined chain.

ACCEPT means that the packet will pass through. DENY means that the packet is not accepted and thrown away without any form of notification to the sender. MASQ means that the packet will be masqueraded if it came from the localhost or demasqueraded if it is a response to an earlier sent masqueraded packet. MASQ is only allowed for the forward chain and the user-defined chains. REDIRECT will redirect packets to a local port. REDIRECT is only allowed for the input chain and for user-defined chains. REJECT means that the packet is not accepted and that the sender will be notified of this fact by means of an ICMP message. RETURN has the same effect as a packet reaching the end of a user-defined chain without match, i.e., the next rule in the previous chain will be checked. If the end of a built-in chain is reached, or a rule in a built-in chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet.

Descriptions of the various stages and their relationships:

Checksum

The checksum is calculated on the header of the IP packet. If the checksum is incorrect, the packet is denied. The checksum field is the 16-bit one's complement of the one's complement sum of all 16-bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero. See RFC791 for more information.

Sanity

There is a sanity check before each firewall chain, but the check before the input chain is the most important. Some malformed packets might confuse the rule-checking code, and these are denied here (a message is printed to the syslog if this happens).

INPUT

Let's say a packet arrives at eth0, has a correct checksum, and the sanity is ok too. The packet then enters the INPUT chain and is accepted, rejected or denied. The difference between deny and reject lies in the response sent to the sender of the packet. In case of deny, the sender is never notified, in case of reject, the sender is notified if the packet was not an ICMP packet.

Demasquerade

In case masquerading is used, this is the step where the opposite is done. Afterward, the demasqueraded packet goes directly to the OUTPUT chain. If masquerading isn't used, the packet goes into the routing decision process.

Routing decision

Here it is decided, based on the destination of the packet, if the packet should go to a local process on the same machine or to a process on another machine, in which case the packet is sent to the FORWARD chain.

Local Process

This can be any local process that can accept and send packets.

FORWARD

All packets arriving at this machine that are destined for another machine go through the FORWARD chain and are accepted, denied or rejected.

OUTPUT

All packets that go out an interface pass through the OUTPUT chain and are accepted, denied or rejected.

The Firm's network with IPCHAINS

The best way to secure things is to define what is allowed, consider everything else to be not allowed and see to it that what is not allowed is not possible.

How does this reflect on the usage of IPCHAINS ? Before explaining things, let's make a few functional assumptions in relation to the Firm's network:

- We are running a nameserver on Machine 4
- Packets from Machines 1-3 aimed at the outside world are masqueraded by Machine 4 and forwarded.
- We can **ping** machines on our local net as well as machines in the outside world, but we ourselves do not respond to any pings from the outside world, i.e., arriving at our eth1 interface.
- We are participating in a distributed.net project: rc564 cracking. We are running a proxy on Machine 4

that communicates with a server on the Internet. Machines 1-3 get their data from the proxy server and send their computed results to the proxy server.

- Of course we also want to be able to view web-pages with http and https.

First, we delete all rules for all built-in chains:

```
# ipchains -F input
# ipchains -F forward
# ipchains -F output
```

Then we delete all user-defined chains:

```
# ipchains -X
```

The we tell **ipchains** to DENY all traffic:

```
# ipchains -P input DENY
# ipchains -P forward DENY
# ipchains -P output DENY
```

The next thing to do is allow traffic on a need basis. Let's start with forwarding:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

We trust everything coming from the local Machines 1-3. Machine 4 receives this on its eth0 interface (192.168.0.1) and thus the **ipchains** commands needed are:

```
# ipchains -A input -i eth0 -j ACCEPT
# ipchains -A output -i eth0 -j ACCEPT
```

The same goes for the lo interface. Without this definition, certain services, such as a caching DNS Server, will not function correctly:

```
# ipchains -A input -i lo -j ACCEPT
# ipchains -A output -i lo -j ACCEPT
```

We want to be able to view web-pages. This is done via either the http or the https protocol which use port 80 and port 443 respectively:

```
# ipchains -A output -i eth1 -p tcp --dport 80  -j ACCEPT
# ipchains -A input  -i eth1 -p tcp --sport 80  -j ACCEPT ! -y
# ipchains -A output -i eth1 -p tcp --dport 443 -j ACCEPT
# ipchains -A input  -i eth1 -p tcp --sport 443 -j ACCEPT ! -y
# ipchains -A forward -p tcp -i eth1 -s 192.168.0.0/24 --dport 80 -j MASQ
# ipchains -A forward -p tcp -i eth1 -s 192.168.0.0/24 --dport 443 -j MASQ
```

We also want to enable our DNS, running on Machine 4, to do requests on other nameservers, DNS uses port 53 for sending the queries and for receiving the answers to the queries:

```
# ipchains -A output -i eth1 -p udp --dport 53 -j ACCEPT
# ipchains -A input  -i eth1 -p udp --sport 53 -j ACCEPT
```

For the rc564 proxyserver running on Machine 4 which uses port 2064 to receive new and send computed

keys, we need the following two lines:

```
# ipchains -A output -i eth1 -p tcp --dport 2064 -j ACCEPT
# ipchains -A input  -i eth1 -p tcp --sport 2064 -j ACCEPT ! -y
```

We want to be able to ping hosts on the Internet. This is done with so-called ICMP packets. ICMP stands for Internet Control Message Protocol. ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram or when the gateway can direct the host to send traffic on a shorter route. There are several ICMP types. Typing **ipchains -h icmp** will show you which types are valid:

```
Type Code Description
0   0    echo-reply (pong)
3        destination-unreachable
   0     network-unreachable
   1     host-unreachable
   2     protocol-unreachable
   3     port-unreachable
   4     fragmentation-needed
   5     source-route-failed
   6     network-unknown
   7     host-unknown
   9     network-prohibited
   10    host-prohibited
   11    TOS-network-unreachable
   12    TOS-host-unreachable
   13    communication-prohibited
   14    host-precedence-violation
   15    precedence-cutoff
4   0    source-quench
5        redirect
   0     network-redirect
   1     host-redirect
   2     TOS-network-redirect
   3     TOS-host-redirect
8   0    echo-request (ping)
9   0    router-advertisement
10  0    router-solicitation
11       time-exceeded (ttl-exceeded)
   0     ttl-zero-during-transit
   1     ttl-zero-during-reassembly
12       parameter-problem
   0     ip-header-bad
   1     required-option-missing
13  0    timestamp-request
14  0    timestamp-reply
17  0    address-mask-request
18  0    address-mask-reply
```

Because we want to be able to ping hosts on the Internet, we need to allow outgoing echo-request or ping. Since we also want to be able to receive the answers to a ping, we must allow incoming echo-reply or pong:

```
# ipchains -A output -i eth1 -p icmp --icmp-type ping -j ACCEPT
# ipchains -A input  -i eth1 -p icmp --icmp-type pong -j ACCEPT
```

We also allow incoming and outgoing destination-unreachable ICMP messages on both the internal (eth0) interface and the external (eth1) interface. These occur if a datagram can't be delivered to its destination.

```
# ipchains -A input  -p icmp --icmp-type destination-unreachable -j ACCEPT
# ipchains -A output -p icmp --icmp-type destination-unreachable -j ACCEPT
```

If a gateway hasn't got the buffer space needed to queue the datagrams for output to the next network on the route to the destination network, it discards Internet datagrams and sends source-squench messages. On receipt of a source-quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source-quench messages from the gateway.

So we must also allow incoming and outgoing source-quench messages on both the internal (eth0) interface and the external (eth1) interface.

```
# ipchains -A input  -p icmp --icmp-type source-quench -j ACCEPT
# ipchains -A output -p icmp --icmp-type source-quench -j ACCEPT
```

We do not allow redirect messages because this is not applicable in our situation. This kind of message is used to tell a host that there is a shorter route to a network via another gateway. We only have one gateway, Machine 4, so there is no need for it.

We do not allow router-advertisement and router-solicitation messages. These messages enable hosts attached to multicast or broadcast networks to discover the IP addresses of their neighboring routers. For more information, consult RFC1256.

Datagrams contain a field called TTL, which stands for Time To Live. This field is decremented by each gateway the datagram passes through. If a gateway sees that the value of the TTL field is zero, it will discard the datagram and notify the source of this situation by means of an ICMP time-exceeded message. We allow both incoming and outgoing ICMP time-exceeded messages on all interfaces:

```
# ipchains -A input  -p icmp --icmp-type time-exeeded -j ACCEPT
# ipchains -A output -p icmp --icmp-type time-exeeded -j ACCEPT
```

If a datagram is discarded by a host or gateway because it can't be processed due to a problem with the header parameters, the host or gateway will send an ICMP parameter-problem message. We allow both incoming and outgoing ICMP parameter-problem messages on all interfaces:

```
# ipchains -A input  -p icmp --icmp-type parameter-problem -j ACCEPT
# ipchains -A output -p icmp --icmp-type parameter-problem -j ACCEPT
```

We are also not interested in timestamp-request and timestamp-reply messages. For more information, consult RFC792.

A host can use ICMP address-mask-request messages to ask for the address mask of a certain machine. ICMP address-mask-reply messages are used to transmit the answer to the request. See RFC950 for more information if you're interested. We don't allow these type of messages because we don't need them.

Finally, we must allow the masquerading of any ICMP packet. As a result of the rules mentioned above, "any ICMP packet" can only be one of the packets we allow:

```
# ipchains -A forward -p icmp -i eth1 -s 192.168.0.0/24 -j MASQ
```

You don't have to know all icmp-types by heart. I've described them to point out that more happens than meets the eye. These topics are described in-depth in the RFC's 792, 950, and 1256, which are available from multiple sources on the Internet.

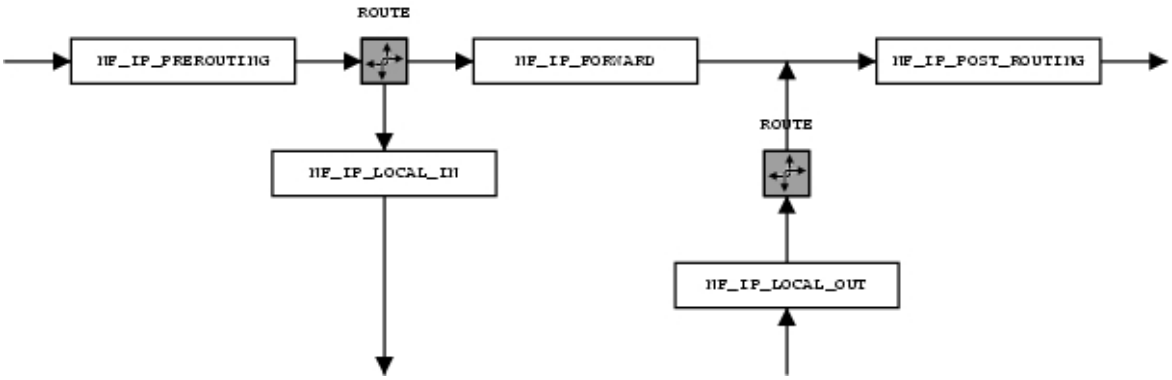
IPTABLES, an overview

What is it?

Linux kernels as of version 2.3.15 support a new framework designed for packet filtering called "netfilter" which must be compiled into the kernel. This is done by selecting "Networking options -->" which brings you to the "Networking options" page. Enabling the option "Network packet filtering" sets CONFIG_NETFILTER to "Y" and adds the "IP: Netfilter Configuration -->" option to the page. Selecting this option takes you to the "IP: Netfilter Configuration" page. You should at least enable the "Connection tracking (CONFIG_IP_NF_CONNTRACK)" and the "IP tables support (CONFIG_IP_NF_IPTABLES)". The latter adds a lot of options from which you should at least enable "Connection state match support (CONFIG_IP_NF_MATCH_STATE)", "Packet Filtering (CONFIG_IP_NF_FILTER)" and "Full NAT (CONFIG_IP_NF_NAT)".

The user space tool **iptables** is used to tell the "netfilter" code in the kernel which packets to filter.

Netfilter "hooks"



As the figure above shows, netfilter supports five different hooks in the protocol stack. Imagine for a moment that packets pass through a tube. A hook is a part in the tube, where an additional piece of tube can be inserted. In this piece of tube works a little man who examines and changes when needed, every passing packet based on his job description, the so called "rules". The little man reports to his boss, netfilter, what to do with the packet.

There are five possible actions:

NF_ACCEPT

Continue traversal as normal.

NF_DROP

Drop the packet and don't continue traversal.

NF_QUEUE

Queue the packet for userspace handling.

NF_REPEAT

Call this hook again.

NF_STOLEN

I've taken over the packet, don't continue traversal.

Tables and Chains

By default five chains and three tables are supported. As the figure below shows, certain chains are only valid for certain tables.

Table 12.1. Valid chains per table

		CHAIN				
		PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING

TABLE	MANGLE	V			V	
	NAT	V			V	V
	FILTER		V	V	V	

The "filter" table

The filter table is used for filtering packets. The filter table contains three chains. The INPUT chain is used for all packets that are for the firewall. The FORWARD chain is used for all packets that come from outside the firewall and are destined for another machine. The OUTPUT chain is used for all packets generated by the firewall.

The "nat" table

The nat table is used for Network Address Translation. The nat table contains three chains. The PREROUTING chain is the first used to alter packets. The OUTPUT chain is used to alter packets generated by the firewall. The POSTROUTING chain is the last chain where packets can be altered as they leave the firewall.

The "mangle" table

The mangle table is used to mangle packets. We can change several things but we can't do masquerading or network address translation here. The mangle table contains two chains. The PREROUTING chain is the first chain alter packets. The OUTPUT chain is used to alter packets generated by the firewall.

Connection tracking: Stateful Firewalling

Firewalls that are able to do connection tracking are called Stateful Firewalls. What it comes down to is that connections are tracked by remembering what (type of) packets have been received and sent.

Incoming packets in response to a **ping**, for instance, can be accepted by the firewall because the firewall knows that we've caused this ourselves by doing the **ping** in the first place.

The **iptables** option used for connection tracking is the "--state" option.

The manual page describes this as follows:

state

This module, when combined with connection tracking, allows access to the connection tracking state for this packet.

--state state

Where state is a comma-separated list of the connection states to match.

Possible states are:

INVALID

The packet is associated with no known connection.

ESTABLISHED

The packet is associated with a connection which has seen packets in both directions.

NEW

The packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions.

RELATED

The packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer or an ICMP error.

There are two modules for connection tracking:

ip_conntrack

The main connection-tracking code.

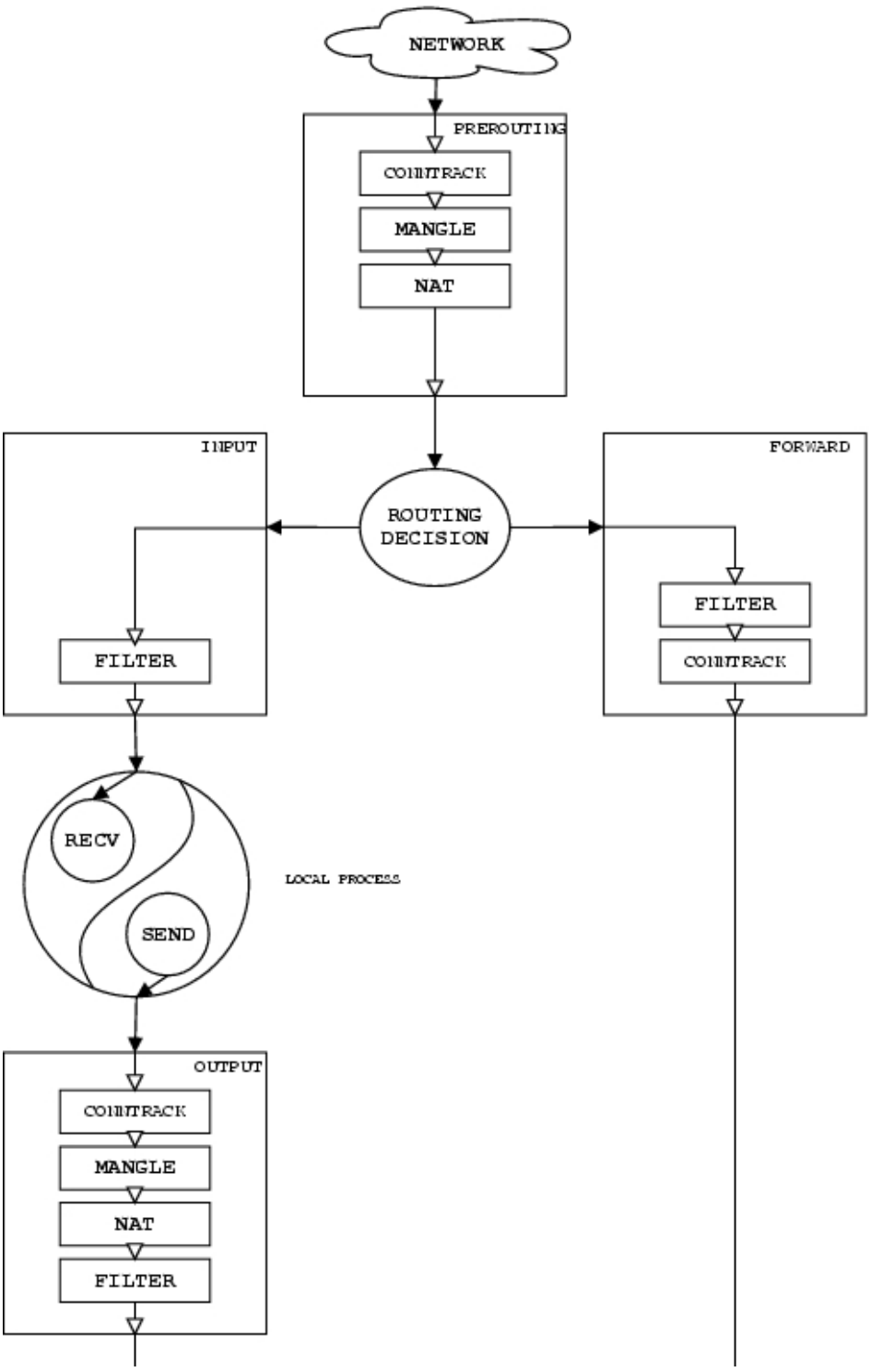
ip_conntrack_ftp

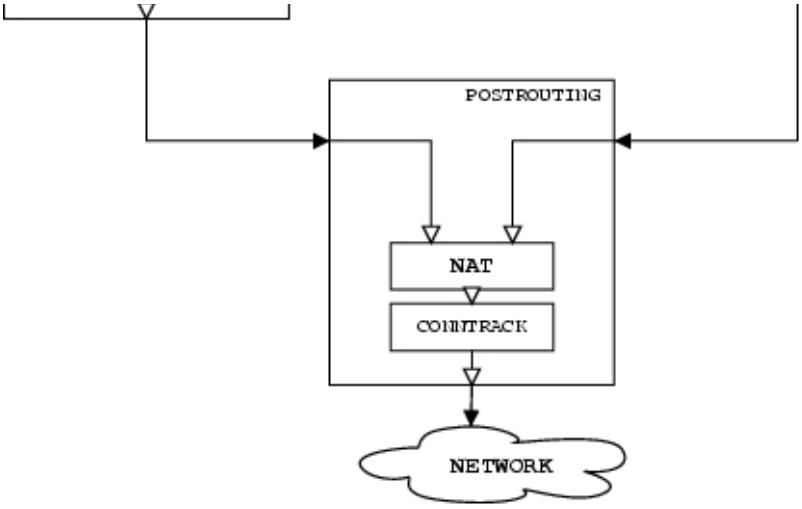
Additional code needed to track ftp connections, both active and passive.

Conntrack hooks in at PREROUTING, FORWARD, OUTPUT and POSTROUTING.

Hooks, Tables and Chains put together

Putting what we've discussed so far into one picture:





Adding extra functionality

Adding targets

Each rule specifies what to do with a packet matching the rule. The "what-to-do" part is called the target. Standard targets always present are:

ACCEPT means let the packet through.

DROP means throw the packet on the floor

QUEUE means pass the packet to user space.

RETURN means stop traversing this chain and resume at the next rule in the previous calling chain. If the end of a built-in chain is reached or a rule in a built-in chain with target RETURN matches the packet, the target specified in the chain policy determines the fate of the packet.

Included in the standard distribution are a number of target extensions for which support in the kernel must be enabled if you wish to use them. Consult the man page of **iptables** for further details. Most of these targets have options. The extension LOG for instance, has the following five options: "--log-level", "--log-prefix", "--log-tcp-sequence", "--log-tcp-options", "--log-ip-options". Please consult the man page for details on options per target.

The extended target modules included in the distribution are:

LOG

Turn on kernel logging of matching packets. When this option is set for a rule, the Linux kernel will print some information on all matching packets (such as most IP header fields) via printk().

MARK

This is used to set the netfilter mark value associated with the packet. It is only valid in the mangle table.

REJECT

This is used to send back an error packet in response to the matched packet; otherwise, it is equivalent to DROP. This target is only valid in the INPUT, FORWARD and OUTPUT chains and user-defined chains which are only called by those chains.

TOS

This is used to set the 8-bit Type of Service field in the IP header. It is only valid in the mangle table.

MIRROR

This is an experimental demonstration target which inverts the source and destination fields in the IP header and retransmits the packet. It is only valid in the INPUT, FORWARD and OUTPUT chains and user-defined chains which are only called by those chains.

SNAT

This target is only valid in the the POSTROUTING chain of the nat table. It specifies that the source address of the packet should be modified (and all future packets in this connection will also be mangled), and rules should cease being examined.

DNAT

This target is only valid in the PREROUTING, OUTPUT and user-defined chains (which are only called by those chains) of the nat table. It specifies that the destination address of the packet should be modified (and all future packets in this connection will also be mangled), and rules should cease being examined.

MASQUERADE

This target is only valid in the POSTROUTING chain of the nat table. It should only be used with dynamically assigned IP (dialup) connections: if you have a static IP address, you should use the SNAT target. Masquerading is equivalent to specifying a mapping to the IP address of the interface the packet is going out, but also has the effect that connections are forgotten when the interface goes down. This is the correct behaviour when the next dialup is unlikely to have the same interface address (and hence any established connections are lost anyway).

REDIRECT

This target is only valid in the PREROUTING, OUTPUT and user-defined chains (which are only called by those chains) of the nat table. It alters the destination IP address to send the packet to the machine itself (locally-generated packets are mapped to the 127.0.0.1 address).

Adding matching modules

Each rule specifies what to do with a packet matching that rule. The "match" part is implemented by packet matching modules. Most of these modules support options. Please consult the man page for detailed information on the options.

The following modules are included in the distribution:

tcp

These extensions are loaded if "--protocol tcp" is specified, and no other match is specified.

udp

These extensions are loaded if "--protocol udp" is specified, and no other match is specified.

icmp

This extension is loaded if "--protocol icmp" is specified, and no other match is specified.

mac

Match source MAC address. It must be of the form XX:XX:XX:XX:XX:XX. Note that this only makes sense for packets entering the PREROUTING, FORWARD or INPUT chains for packets coming from an ethernet device.

limit

This module matches at a limited rate using a token bucket filter: it can be used in combination with the LOG target to give limited logging. A rule using this extension will match until this limit is reached (unless the "!" flag is used).

multiport

This module matches a set of source or destination ports. Up to 15 ports can be specified. It can only be used in conjunction with -p tcp or -p udp.

mark

This module matches the netfilter mark field associated with a packet (which can be set using the MARK target).

owner

This module attempts to match various characteristics of the packet creator for locally-generated packets. It is only valid in the OUTPUT chain, and even then some packets (such as ICMP ping responses) may have no owner and hence, never match.

state

This module, when combined with connection tracking, allows access to the connection tracking state for this packet.

unclean

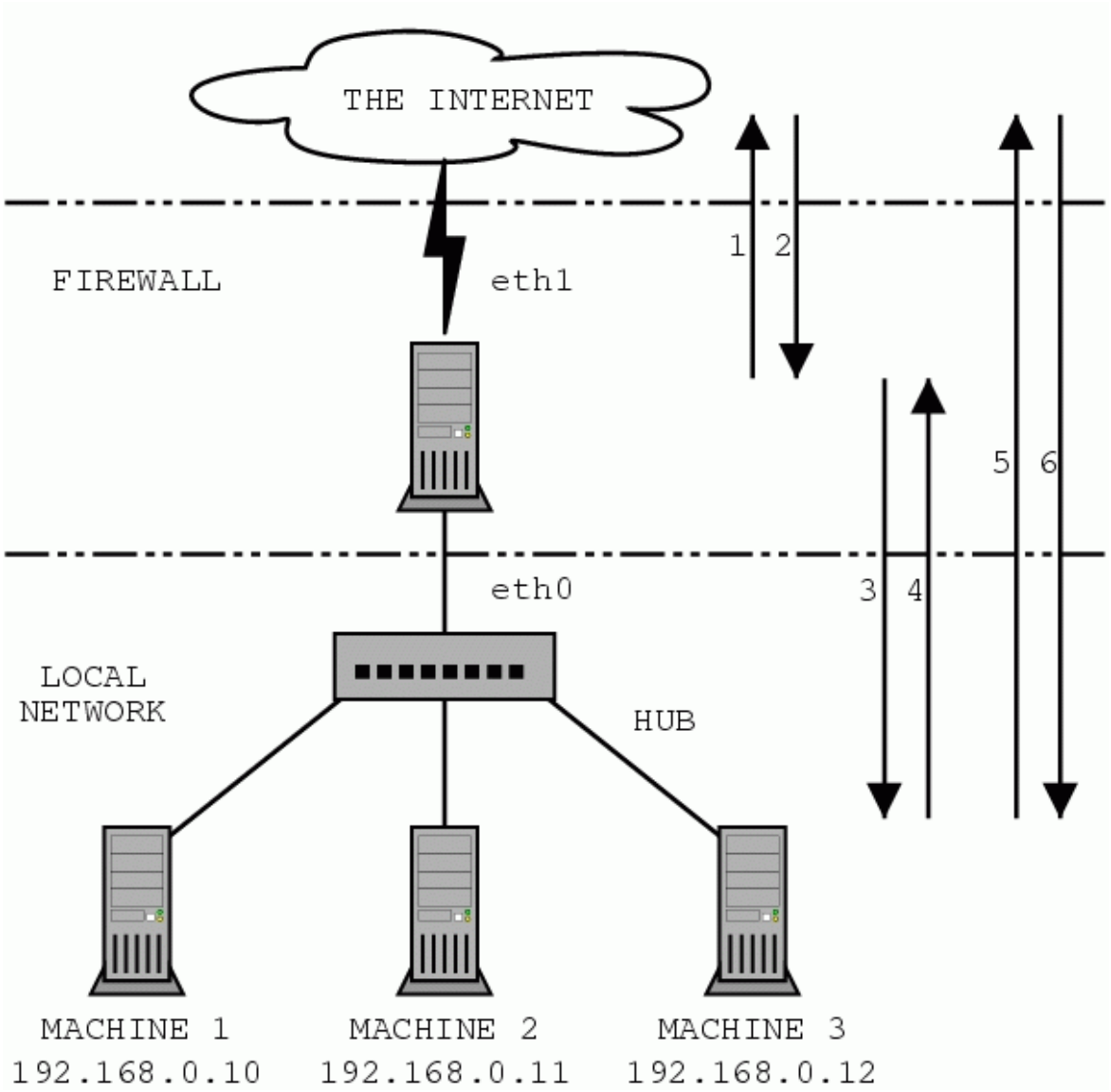
This module takes no options, but attempts to match packets which seem malformed or unusual. This is regarded as experimental.

tos

This module matches the 8 bits of Type of Service field in the IP header (ie. including the precedence bits).

The Firm's network with IPTABLES

We all remember The Firm's network from the previous section. Let's visit it again. The picture below shows The Firm's network and the possible combinations of traffic initiation/destination.



(1) Traffic initiated by the Firewall that is destined for the Internet

We are running a DNS on the Firewall that needs to be able to consult other DNSes on the Internet (which use the UDP protocol and listen to PORT 53). We want to be able to use **ssh** (which uses the TCP protocol and port 22) to connect to other systems on the Internet. We are participating in a distributed.net project RC564 cracking and are running a proxy server on the Firewall (which uses the TCP protocol and PORT 2064 to communicate with the keyserver). We want to be able to **ping** hosts on the Internet (**ping** uses the

ICMP protocol and message type "ping"). The Firewall communicates with the Internet through interface eth1. Taking all this into consideration, the **iptables** commands we need are:

```
iptables -t filter -A OUTPUT -o eth1 -p udp  --destination-port dns       \
-m state --state NEW -j ACCEPT
iptables -t filter -A OUTPUT -o eth1 -p tcp  --destination-port ssh       \
-m state --state NEW -j ACCEPT
iptables -t filter -A OUTPUT -o eth1 -p tcp  --destination-port 2064      \
-m state --state NEW -j ACCEPT
iptables -t filter -A OUTPUT -o eth1 -p icmp --icmp-type ping             \
-m state --state NEW -j ACCEPT
```

These four **iptables** commands tell the firewall to allow outgoing connection-initialization packets for dns, ssh, RC564 cracking and ping.

(2) Traffic initiated by the Internet that is destined for the Firewall

We want to be able to use **ssh**, which uses the TCP protocol and port 22, to connect to our Firewall from other systems on the Internet. The **iptables** commands we need are:

```
iptables -t filter -A INPUT  -i eth1 -p tcp --destination-port ssh        \
-m state --state NEW -j ACCEPT
```

This **iptables** command tells the firewall to allow incoming connection initialization packets for ssh.

(3) Traffic initiated by the Firewall that is destined for the internal network

We want to be able to use **ssh**, which uses the TCP protocol and port 22, to connect to one of our internal machines from our Firewall. The **iptables** commands we need are:

```
iptables -t filter -A OUTPUT -o eth0 -p tcp --destination-port ssh        \
-m state --state NEW -j ACCEPT
```

This **iptables** command tells the Firewall to allow outgoing SSH connection initialization packets destined for a machine on the Internal Network.

(4) Traffic initiated by the internal network that is destined for the firewall

The machines on the internal network, using the dns of the firewall, must be able to connect to the firewall using ssh, are processing RC564 keys, must be able to talk to the proxy on the firewall using port 2064 and must be able to ping the Firewall for system administrative purposes. The **iptables** commands we need are:

```
iptables -t filter -A INPUT  -i eth0 -p udp --destination-port dns        \
-m state --state NEW -j ACCEPT
iptables -t filter -A INPUT  -i eth0 -p tcp --destination-port ssh        \
-m state --state NEW -j ACCEPT
iptables -t filter -A INPUT  -i eth0 -p tcp --destination-port 2064       \
-m state --state NEW -j ACCEPT
iptables -t filter -A INPUT  -i eth0 -p icmp --icmp-type ping             \
-m state --state NEW -j ACCEPT
```

These four **iptables** commands tell the Firewall to allow incoming connection-initialization packets for dns, ssh, RC564 cracking and ping.

(5) Traffic initiated by the Internal Network that is destined for the Internet

Every connection from a machine on the internal network to a machine on the Internet is allowed. The

iptables commands we need are:

```
iptables -t filter -A FORWARD -i eth0 -o eth1 -m state --state NEW -j ACCEPT
```

This **iptables** command tells the Firewall to allow ALL outgoing connection initialization packets.

(6) Traffic initiated by the Internet that is destined for the Internal Network

This does not occur because our local network uses private IP addresses that can't be used on the Internet. Our local machines aren't visible from the Internet.

What we could do to make one of our machines available on the Internet is to let people connect to a certain port on the firewall and use NAT to redirect them to a port on one of the machines on the Internal Network.

Suppose Machine 2 has a web-server (or some other program) running which listens to port 2345 and people from the outside must be able to connect to that program. Since The Firm is using private IP addresses for their Internal Network, Machine 2 is not visible on the Internet. The solution here is to tell Machine 4 that all data from the outside that is aimed at port 80 should be routed to port 2345 on Machine 2. The **iptables** commands we need are:

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --destination-port 80       \
-j DNAT --to-destination 192.168.0.11:2345
iptables -t filter -A FORWARD -i eth1 -p tcp --destination-port 2345     \
-m state --state NEW -j ACCEPT
```

The first line changes the destination address and port. Since this then becomes traffic aimed at another machine, the traffic must pass the FORWARD filter. The second line sees to it that that happens.

(!) Traffic as a result of initiated traffic

So far, we've only specified that connection initiation traffic is allowed, but that is not enough. We also must allow ESTABLISHED and RELATED traffic.

Let's tell the firewall that all ESTABLISHED and RELATED traffic, regardless of type, interface etc. is allowed. We must also allow the initiation of traffic on the firewalls lo interface because otherwise some services, such a a caching DNS server, will not work. We need the following **iptables** commands to realize this:

```
iptables -t filter -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A INPUT   -m state --state NEW -i lo          -j ACCEPT
```

Remember that this can't be a problem because the packets are a result of the fact that we've accepted the initiation of the connection in the first place.

All iptables commands put together

Adding stuff to start with a clean sheet and telling the firewall to masquerade packets from the internal network aimed at the Internet can be accomplished with the following commands:

```
#############################################################################
# FLUSH ALL RULES IN THE MANGLE, NAT AND FILTER TABLES
#############################################################################
iptables -t mangle -F
iptables -t nat    -F
iptables -t filter -F
```

```
#########################################################################
# DELETE ALL USER-DEFINED (NOT BUILT-IN) CHAINS IN THE TABLES
#########################################################################
iptables -t mangle -X
iptables -t nat    -X
iptables -t filter -X

#########################################################################
# SET ALL POLICIES FOR ALL BUIL-IN CHAINS TO DROP
#########################################################################
iptables -P INPUT   DROP
iptables -P FORWARD DROP
iptables -P OUTPUT  DROP

#########################################################################
# (1) TRAFFIC INITIATED BY THE FIREWALL DESTINED FOR THE INTERNET
#     DNS, SSH, RC564, PING
#########################################################################
# ALLOW INITIATION BY THE FIREWALL
iptables -t filter -A OUTPUT -o eth1 -p udp  --destination-port dns       \
-m state --state NEW -j ACCEPT
iptables -t filter -A OUTPUT -o eth1 -p tcp  --destination-port ssh       \
-m state --state NEW -j ACCEPT
iptables -t filter -A OUTPUT -o eth1 -p tcp  --destination-port 2064      \
-m state --state NEW -j ACCEPT
iptables -t filter -A OUTPUT -o eth1 -p icmp --icmp-type ping             \
-m state --state NEW -j ACCEPT
# ALLOW INCOMING RESPONSES
iptables -t filter -A INPUT  -i eth1                                      \
-m state --state ESTABLISHED,RELATED -j ACCEPT

#########################################################################
# (2) TRAFFIC INITIATED BY THE OUTSIDE DESTINED FOR THE FIREWALL
#     SSH
#########################################################################
# ALLOW INITIATION
iptables -t filter -A INPUT  -i eth1 -p tcp  --destination-port ssh       \
-m state --state NEW -j ACCEPT
# ALLOW RESPONSE
iptables -t filter -A OUTPUT -o eth1 -p tcp  --destination-port ssh       \
-m state --state ESTABLISHED,RELATED -j ACCEPT

#########################################################################
# (3) TRAFFIC INITIATED BY THE FIREWALL DESTINED FOR THE INTERNAL NETWORK
#     SSH
#########################################################################
# ALLOW INITIATION
iptables -t filter -A OUTPUT -o eth0 -p tcp  --destination-port ssh       \
-m state --state NEW -j ACCEPT
# ALLOW RESPONSE
iptables -t filter -A INPUT  -i eth0 -p tcp  --destination-port ssh       \
-m state --state ESTABLISHED,RELATED -j ACCEPT

#########################################################################
# (4) TRAFFIC INITIATED BY THE INTERNAL NETWORK DESTINED FOR THE FIREWALL
#     DNS, SSH, RC564, PING
#########################################################################
# ALLOW INITIATION
iptables -t filter -A INPUT  -i eth0 -p udp  --destination-port dns       \
-m state --state NEW -j ACCEPT
```

```
iptables -t filter -A INPUT  -i eth0 -p tcp  --destination-port ssh       \
-m state --state NEW -j ACCEPT
iptables -t filter -A INPUT  -i eth0 -p tcp  --destination-port 2064      \
-m state --state NEW -j ACCEPT
iptables -t filter -A INPUT  -i eth0 -p icmp --icmp-type ping             \
-m state --state NEW -j ACCEPT
# ALLOW RESPONSE
iptables -t filter -A OUTPUT -o eth0                                      \
-m state --state ESTABLISHED,RELATED -j ACCEPT

##############################################################################
# (5) TRAFFIC INITIATED BY THE INTERNAL NETWORK DESTINED FOR THE OUTSIDE
#     EVERYTHING WE CAN INITIATE IS ALLOWED
##############################################################################
# ALLOW INITIATION OF EVERYTHING
iptables -t filter -A FORWARD -i eth0 -o eth1                             \
-m state --state NEW -j ACCEPT
# ALLOW RECEPTION
iptables -t filter -A FORWARD -i eth1 -o eth0                             \
-m state --state ESTABLISHED,RELATED -j ACCEPT

##############################################################################
# (6) TRAFFIC INITIATED BY THE OUTSIDE DESTINED FOR THE INTERNAL NETWORK
#     ALL FORBIDDEN, EXCEPT WEBSERVER FORWARDING TO INTERNAL MACHINE
##############################################################################
# ALLOW DESTINATION NAT FROM FIREWALL:80 TO INTERNAL MACHINE:2345
iptables -t nat -A PREROUTING -i eth1 -p tcp --destination-port 80        \
-j DNAT --to-destination 192.168.0.11:2345
iptables -t filter -A FORWARD -i eth1 -p tcp --destination-port 2345      \
-m state --state NEW -j ACCEPT

##############################################################################
# (!) TRAFFIC AS A RESULT OF INITIATED TRAFFIC
#     ALL ALLOWED
##############################################################################
# ALLOW ALL PACKETS RESULTING FROM ALLOWED CONNECTIONS
iptables -t filter -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A INPUT   -m state --state NEW -i lo          -j ACCEPT

##############################################################################
# MASQUERADE PACKAGES FROM OUR INTERNAL NETWORK DESTINED FOR THE INTERNET
# THIS IS SNAT (SOURCE NAT)
##############################################################################
iptables -t nat    -A POSTROUTING -i eth0 -o eth1 -s 192.168.0.0/24 -j MASQUERADE

##############################################################################
# ENABLE FORWARDING
##############################################################################
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Saving And Restoring Firewall Rules

Firewall rules can be saved and restored easily by using the commands **ipchains-save**, which writes to stdout and **ipchains-restore**, which reads from stdin. Assuming we use the file fwrules.saved to save/restore the rules, the two commands are:

```
ipchains-save > fwrules.saved
ipchains-restore < fwrules.saved
```

Similar commands exist for **iptables**:

```
iptables-save > fwrules.saved
iptables-restore < fwrules.saved
```

Denial of Service (DOS) attacks

What they are

DoS attackers misuse the fact that resources on the Internet are limited and that services can be disrupted by taking away (one of) their resources: storage-capacity, bandwith or processor-capacity. This is done by "packet flooding".

Packet flooding can be done with TCP, ICMP and UDP.

When using TCP mostly the SYN, ACK and RST flags are used.

When using ICMP, the message types echo request and echo reply are used. This is called "ping-flooding".

When using UDP, the chargen and echo UDP services are used.

Also, two systems can be played out against each other by a third system. The third system changes the source IP-address of the packages it sends to the first system to that of the second system. The first system then thinks the packets came from the second system and starts sending replies to the second system. This method is called "DoS with IP address spoofing".

Check the site http://www.cert.org/ for a complete history of DoS and DDos (Distributed DoS) attacks. And have a look at RFC2827 which describes Network Ingress Filtering, a method to prevent IP address spoofing. This doesn't prevent DoS attacks but makes it possible to trace the real IP address of the offender.

What to do against them

It is not possible to fully prevent DoS attacks without disconnecting from the Internet. What you can do to minimize the effects of DoS and DDoS attacks is apply some packet filtering and rate limiting rules to the firewall.

Routed

What it is

routed is a program that can automatically adjust routing tables based on changes in the network.

When to use it

If there are multiple possible paths to a certain destination, and you want an alternate route to that destination to be selected automatically (in case the default route to that destination is not usable for some reason) the **routed** program can do this for you automatically.

PortSentry: Preventing port scans

Description

PortSentry is part of the Abacus Project suite of security tools. It is a program designed to detect and respond to port scans against a target host in real-time. Some of its more useful features include:

- Runs on TCP and UDP sockets to detect port scans against your system. PortSentry is configurable to run on multiple sockets at the same time so you only need to start one copy to cover dozens of tripwired services.
- Stealth scan detection (Linux only right now). PortSentry will detect SYN/half-open, FIN, NULL, X-MAS

and oddball packet stealth scans. Four stealth scan operation modes are available for you to choose from.
- PortSentry will react to a port scan attempt by blocking the host in real-time. This is done through configured options of either dropping the local route back to the attacker, using the Linux ipfwadm/ ipchains command, *BSD ipfw command and/or dropping the attacker host IP into a TCP Wrappers hosts. deny file automatically.
- PortSentry has an internal state engine to remember hosts that connected previously. This allows the setting of a trigger value to prevent false alarms and detect "random" port probing.
- PortSentry will report all violations to the local or remote syslog daemons indicating the system name, time of attack, attacking host IP and the TCP or UDP port a connection attempt was made to. When used in conjunction with Logcheck it will provide an alert to administrators through e-mail.
- Once a scan is detected your system will turn into a black hole and disappear from the attacker. This feature stops most attacks cold.

Installation and Configuration

After you've downloaded the correct version, and untarred it, you've got a directory portsentry-x.y which contains the portsentry files and some README files. The information below comes from those README files.

The first thing to do is verify (and change if appliccable) the contents of the portsentry_config.h file:

- CONFIG_FILE - The path to the PortSentry configuration file.
- WRAPPER_HOSTS_DENY - The path and name of TCP wrapper hosts.deny file.
- SYSLOG_FACILITY - The syslog facility for PortSentry to use.
- SYSLOG_LEVEL - The syslog level to send messages.

The second thing to do is verify (and change if appliccable) the contents of the portsentry.conf file:

- TCP_PORTS - A comma delimited string of TCP ports you want PortSentry to listen to. This string CANNOT have any spaces in it. You can list as many sockets as you would like. PortSentry will try to bind them all up until the default limit of 64.
- UDP_PORTS - The same as above, except for UDP ports. You need to be very careful with UDP mode as an attacker can forge a port sweep and make you block any number of hosts. Use this option with caution or not at all if your host is a well-known Internet connected system.
- ADVANCED_PORTS_TCP - A value indicating the highest port number to monitor down from. Any port *below* this number is then monitored. The default is 1024 (reserved port range), but can be as large as 65535 (system max). I don't recommend going over 1024 with this option.
- ADVANCED_PORTS_UDP - Same as above, except for UDP.
- ADVANCED_EXCLUDE_TCP - A comma delimited string of TCP ports that should be manually excluded from monitoring in Advanced mode. These are normally ports that may get hit by mistake by remote clients and shouldn't cause alarms (ident, SSL, etc).
- ADVANCED_EXCLUDE_UDP - Same as above, except for UDP.
- IGNORE_FILE - The path to the file that contains IP addresses of hosts you want to always be ignored. This will be explained later.
- BLOCKED_FILE - The path to the file that contains the IP addresses of blocked hosts.
- RESOLVE_HOST - This option turns off DNS resolution for hosts. If you have a slow DNS server it may be more effective to turn off resolution.
- BLOCK_UDP - This option disables all automatic responses to UDP probes. Because UDP can be easily forged, it may allow an attacker to start a denial of service attack against the protected host, causing it to block all manner of hosts that should normally be left alone. Setting this option to "0" will disable all responses, although the connects are still logged. This option is mainly useful for Internet exposed hosts. For internal hosts you should leave this enabled. If someone internally is firing spoofed packets at you, then you have a much bigger problem than a denial of service.
- BLOCK_TCP - Same as above, but for TCP. Packet forgery is not as big a problem though because PortSentry waits for a full connect to occur and this is much harder to forge in the basic modes. Leave this enabled, even for Internet connected hosts. For stealth-scan detection modes, the UDP warning applies: Using packet forgery, an attacker can cause you to incorrectly block hosts.
- KILL_ROUTE - This is the command to drop the offending route if an attack is detected. This is the *full path* to the route command along with the necessary parameters to make the command work. The macro $TARGET $ will be substituted with the attacking host IP and is REQUIRED in this option. Your gateway should be a *dead host* on the local subnet. On some systems, you can just use the localhost address (127.0.0.1) and it will probably work. All packets from the target host will be routed to this address, so be careful!. More modern route commands will include a "-blackhole" or "-reject" flag. Check the man pages and, if your route command supports this feature, you should use it (although we recommend using packet filtering instead, see below).
- KILL_HOSTS_DENY - This is the format of the string to drop into the hosts.deny file that TCP wrapper uses. Again the $TARGET$ macro is expanded out to be the IP of the attacker and is required. You can also drop

in any TCP wrapper escape codes (%h, twist, etc). The macro $PORT$ will substitute the port hit by the attacker, but this is NOT required for this option. The macro $MODE$ reports which mode the blocking occurred (tcp, udp, stcp, sudp, atcp, audp), but is also NOT required.

- KILL_RUN_CMD - This is a command to run *before* the route is dropped to the attacker. You can put in any program/script you want executed when an attack is detected. *Putting in retaliatory actions against an attacking host is not recommended* . Virtually every time you're are port scanned, the host doing the scanning has been compromised itself. Therefore, if you retaliate, you are probably attacking an innocent party. Also the goal of security is to make the person GO AWAY. You don't want to irritate them into making a personal vendetta against you. Remember, even a 13 year old can run a [insert favorite D.O.S. program here] attack against you from their Windows box to make your life miserable. As above, the $TARGET$, $PORT$ and $MODE$ macros are available to you, but they are not required with this option.
- KILL_RUN_CMD_FIRST - Setting this to "0" tells the command above to run before the route is dropped. Setting it to "1" makes the command run after the blocking has occurred.
- SCAN_TRIGGER - PortSentry has a state engine that will remember hosts that have connected to it. Setting this value will tell PortSentry to allow X number of grace port hits before it reacts. This will detect both sequential and random port sweeps. The default is 0 which will react immediately. A setting of 1 or 2 will reduce false alarms, anything higher is probably too much as anything more than 3 hits to different ports is pretty suspicious behavior. Usually you can leave this at 0 without any problem except in Advanced stealth-scan detection modes where you may create a "hair trigger" if you aren't careful. Use your own discretion.
- PORT_BANNER - A text banner you want displayed to the connecting host if the PortSentry is activated. Leave this commented out if you don't want this feature. If you do use it, try not to taunt the person too much. It is best to keep it professional and to the point. The banner is *not* displayed when stealth scan detection modes are used.

The third thing to do is to pull the portsentry.ignore file into your editor and add any host you want ignored if it connects to a tripwired port. This should always contain at least the localhost (127.0.0.1) and the IP's of the local interfaces. It is *not* recommended to put in every machine IP on your network, but you can use a netmask to do this. The format for this is:

<IP Address>/<Netmask Bits>

192.168.2.0/24
192.168.0.0/16
etc.

It is not recommended to ignore too much. It may be important for you to see who is connecting to you, even if it is a "friendly" machine. This can help you detect internal host compromises faster. To answer your question: yes, this does happen, and there have been cases of administrators ignoring too much and getting hacked by their own machines as a result.

The fourth thing to do is compile the package. Type **make** and pick your system type and allow it to build and install. The default directory is /usr/local/psionic/portsentry. If you don't like this directory just edit the Makefile and make sure your portsentry.conf and portsentry_config.h files reflect the new path.

Type **make install** after the build to have it copy files to your install directory.

The fifth thing to do is start up PortSentry. PortSentry has six modes of operation. *Only one protocol mode type can be started at a time* (i.e., one TCP mode and one UDP mode). The available modes are:

- **portsentry -tcp** (basic port-bound TCP mode). PortSentry will check the config files and then bind to all TCP ports in the background. To check the init status, just look in the local syslog file to which messages are sent.
- **portsentry -udp** (basic port-bound UDP mode). PortSentry will check the config files and then bind to all UDP ports in the background. If you want to check the init status you, just look in the local syslog to which messages are sent. UDP/Stealth scan warnings apply (read: README.stealth).
- **portsentry -stcp** (Stealth TCP scan detection). PortSentry will use a raw socket to monitor all incoming packets. If an incoming packet is destined for a monitored port it will react to block the host. This method will detect connect() scans, SYN/half-open scans and FIN scans. UDP/Stealth scan warnings apply (read: README.stealth).
- **portsentry -atcp** (Advanced TCP stealth scan detection). PortSentry will start by making a list of all the ports listening in the port area under the ADVANCED_PORTS_TCP option and will then create an exclusion list based on these ports. Any host connecting to *any port* in this range that is *not excluded* (i.e., not a listening network daemon [SMTP, HTTP, etc.]) is blocked. This has some very powerful implications that

you should be aware of: (1) This mode is the m ost sensitive and the most effective of all the protection options. It reacts to port probes with lightning speed because you don't have to wait for them to hit a tripwired port. (2) Because it reacts so abruptly, you may cut off legitimate traffic. An FTP site may send an ident request to you. If you are monitoring the ident port (113 TCP) then you have just cut off the FTP site you were going to! As a result you should put in this list all ports that fall into this situation.

- Advanced Logic Mode - PortSentry is intelligent about how it monitors ports. For some protocols such as FTP, the client actually opens up ports in the ephemeral range (1024-65535) and the server then connects *back* to you. This would normally cause the port scanner to activate. However, PortSentry will look at the incoming connection and determine if it is destined for one of these "temporary" bindings. If it is, then the connection is ignored for that one time. As soon as the connection is torn down the window closes and full protection is back again. This is, in fact, a rudimentary stateful inspection engine. UDP/Stealth scan warnings apply (read: README.stealth).
- **portsentry -sudp** ("Stealth" UDP scan detection). This operates in a manner similar to the TCP stealth mode above. UDP ports need to be listed and are then monitored. This does not bind any sockets, and while not really "stealth" scan detection (doesn't usually apply to UDP), it operates in a similar manner (reacts to *any* UDP packet). UDP/Stealth scan warnings apply (read: README.stealth).
- **portsentry -audp** (Advanced "Stealth" UDP scan detection). This is the same as above except for the UDP protocol. This is a very advanced option and can cause false alarms. This is because PortSentry makes no distinction between broadcast and direct traffic. If you have a router on your local network putting out RIP broadcasts then there is a good chance you will block them. Use this option with extreme caution. You need to be sure to put exclusions into the ADVANCED_EXCLUDE_UDP line (i.e., 520 [RIP]) UDP/Stealth scan warnings apply (read: README.stealth).

Now we're ready to test the installation: Tail the local log and you should see several PortSentry initialization messages. A successful startup looks like this:

```
Oct  9 09:11:35 nemesis portsentry[1644]: adminalert: portsentry is starting.
Oct  9 09:11:36 nemesis portsentry[1644]: adminalert: Going into listen mode on
TCP port: 143
. . .
Oct  9 09:11:37 nemesis portsentry[1644]: adminalert: PortSentry is now active
and listening.
```

The last line indicates the PortSentry is properly initialized, If you don't see this then something has failed.

Securing FTP servers (2.212.3)

The candidate should be able to configure an anonymous download FTP server. This objective includes configuring an FTP server to allow anonymous uploads, listing additional precautions to be taken if anonymous uploads are permitted, configuring guest users and groups with chroot jail, and configuring ftpaccess to deny access to named users or groups.

Key files, terms and utilities include:

ftpaccess, ftpusers, ftpgroups
/etc/passwd
chroot

FTP server Version 6.4/OpenBSD/Linux-ftpd-0.17

Installation

Depending on the Linux distribution you're running, this can be done in several ways (building from sources, using rpm, using apt-get, using aptitude, etc.). Since I'm using the Debian distribution, I've used **apt-get**:

```
# apt-get install ftpd
```

To check which files are installed by this command, type:

```
# dpkg --listfiles ftpd
```

Creating an "ftp" user for anonymous FTP

Anonymous FTP allows users to connect to a system without needing a password. There are two special login names to facilitate this, "anonymous" and "ftp". Both refer to the same account 'ftp' which we'll create, including the home directory and subtree needed:

```
# adduser ftp                 'create user and /home/ftp
....
# chown root.root /home/ftp     'make it owned by root
# chmod 555 /home/ftp          'set it unwritable by anyone, allow subdirs
# cd /home/ftp
# mkdir bin etc lib pub        'create needed sub directories
# chmod 511 bin etc lib        'set it unwritable by anyone
# chmod 555 pub               'set it unwritable by anyone, allow subdirs
# mkdir pub/incoming           'upload directory
```

The "ftp" user will be chroot'ed which means that the root / directory will be remapped to /home/ftp.
The command **ls** is located in the /bin directory which isn't visible anymore. For this reason the **ls** command must

be copied to the /home/ftp/bin directory and the libraries the **ls** command needs and which obviously can't be found either must be copied to the /home/ftp/lib directory:

```
# cd /home/ftp
# cp /bin/ls bin/
# chmod 111 bin/ls              'executable only
```

And now to the libraries the **ls** needs:

```
# ldd /bin/ls
        librt.so.1 => /lib/librt.so.1 (0x4001e000)
        libc.so.6 => /lib/libc.so.6 (0x40030000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x40153000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

We'll copy them:

```
# cp /lib/librt.so.1 lib/
# cp /lib/libc.so.6 lib/
# cp /lib/libpthread.so.0 lib/
# cp /lib/ld-linux.so.2 lib/
# chmod 555 lib/*               'readable and executable
# chown root.root lib/*         'make them all owned by root
```

If we want to present the anonymous user with a message after a succesful login, we can do this by editing the /home/ftp/etc/motd file.

Finally, should we wish to see owner and group names instead of their numbers, the files /etc/passwd and /etc/group must also be copied to the /home/ftp/etc directory. The password field in both files, which is the second field, is not used and should not contain a real password so we'll replace it with an asterisk.

```
# perl -ne 's/^([^:]+:)([^:]*)(:.*)$/\1*\3/; print;' /etc/passwd > etc/passwd
# perl -ne 's/^([^:]+:)([^:]*)(:.*)$/\1*\3/; print;' /etc/group > etc/group
```

Welcome message for all ftp users

When a user connects to the system by means of **ftp**, the contents of the file /etc/ftpwelcome is displayed if that file exists:

```
# ftp pug
Connected to pug.
220- ##########################################################
220- # This is /etc/ftpwelcome                              #
220- #                                                      #
220- #                                                      #
220- #    Registered users must use their loginname and     #
220- #    password to be able to access files in their      #
220- #    home directories.                                 #
220- #                                                      #
```

```
220- ##################################################################
220 pug FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (pug:willem):
```

If the file /etc/ftpwelcome doesn't exist, the following is displayed:

```
# ftp pug
Connected to pug.
220 pug FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (pug:willem):
```

Login message for all not chroot'ed users

After a succesful login, the contents of the file /etc/motd are displayed if the file exists:

```
# ftp pug
Connected to pug.
220- ##################################################################
220- # This is /etc/ftpwelcome                                       #
220- #                                                               #
220- #                                                               #
220- #    Registered users must use their loginname and              #
220- #    password to be able to access files in their               #
220- #    home directories.                                          #
220- #                                                               #
220- ##################################################################
220 pug FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (pug:willem):
331 Password required for willem.
Password:
230-
##################################################################################
230- # This is /etc/motd                                                          #
230- #                                                                           #
230- #    Welcome to our ftp Archive!                                            #
230-
##################################################################################
230 User willem logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Directory specific messages

A directory message is a message that tells something about the contents of the directory. This message, which should be put in the .message file in that directory, only gets displayed when the user changes to that directory during an ftp session:

```
ftp> cd lpic2
250- This directory contains the files for the lpic2 examprep.
```

```
250-
250 CWD command successful.
ftp>
```

Preventing all ftp connections

More accurately: Preventing ALL logins. If the file /etc/nologin exists, logging in to the system is not possible and the contents of the file are displayed:

```
# ftp pug
Connected to pug.
530-
##############################################################################
530- # This is /etc/nologin                                    #
530- #                                                        #
530- # LOGINS ARE NOT PERMITTED AT THE TIME                          #
530- #                                                        #
530- #  please try again later                                  #
530- #                                                        #
530-
##############################################################################
530 System not available.
ftp>
```

Preventing specific users from using ftp

A user whose name is in the file /etc/ftpusers is not allowed ftp access. The user 'root' for instance is not allowed access. Let's try to ftp as root anyway and see what happens:

```
willem@pug:~S ftp pug
Connected to pug.
220- ##############################################################
220- # This is /etc/ftpwelcome                          #
220- #                                                 #
220- #                                                 #
220- #    Registered users must use their loginname and       #
220- #    password to be able to access files in their        #
220- #    home directories.                                #
220- #                                                 #
220- ##############################################################
220 pug FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (pug:willem): root
331 Password required for root.
Password:
530 Login incorrect.
Login failed.
ftp>
```

Restricting specific users to their home directories

A user whose name is in /etc/ftpchroot is treated the same as the "anonymous" or "ftp" user: they're chroot'ed to their home directories, which has the same consequences.

ftpaccess and ftpgroups

The FTP server I've installed, Version 6.4/OpenBSD/Linux-ftpd-0.17, does not use these two files.

The Washington University FTP Server Version wu-2.6.1

Installing

Depending on the Linux distribution you're running this can be done in several ways (building from sources, using rpm, using apt-get, using aptitude etc.). Since I'm using the Debian distribution, I've used **apt-get**:

apt-get install wu-ftpd
Do you want to set up or update an anonymous FTP account now? [n] <y><Enter>

Enter the name of the FTP home directory: [/home/ftp] <Enter>

Do you want to create a directory for user uploads? [n] <y><Enter>

Please look at /etc/wu-ftpd/ftpaccess and its manual page for
further information on how to make /pub/incoming more secure.
Adding system user ftp...
Adding new group ftp (103).
Adding new user ftp (103) with group ftp.
Creating home directory /home/ftp.

Anonymous FTP users will only see UID and GID numbers, instead of
names, because the libnss_files.so library hasn't been installed.

It is not installed by default, since there is no easy way to find
out what version we need to install.

If you want to install it manually, it should be placed in /home/ftp/lib
owned by root, and with permissions of 444 (r--r--r--)

To check which files are installed by this command, type:

dpkg --listfiles wu-ftpd

Creating an "ftp" user for anonymous ftp

This user has been created with the command **addftpuser** during the installation of the server daemon, which includes the accompanying directory structure. The "ftp" user will operate within a chroot'ed environment:

ls -lR /home/ftp
/home/ftp:
total 5
d--x--x--x 2 root root 1024 Dec 6 19:07 bin

```
d--x--x--x   2 root    root        1024 Dec  6 19:07 etc
d--x--x--x   2 root    root        1024 Dec  6 19:07 lib
dr-xr-xr-x   3 root    root        1024 Dec  6 19:11 pub
-rw-r--r--   1 root    root         346 Dec  6 19:07 welcome.msg

/home/ftp/bin:
total 220
---x--x--x   1 root    root       48844 Dec  6 19:07 gzip
---x--x--x   1 root    root       43932 Dec  6 19:07 ls
---x--x--x   1 root    root      128780 Dec  6 19:07 tar

/home/ftp/etc:
total 3
-r--r--r--   1 root    root          18 Dec  6 19:07 group
-r--r--r--   1 root    root          44 Dec  6 19:07 passwd
-r--r--r--   1 root    root         178 Dec  6 19:07 pathmsg

/home/ftp/lib:
total 1374
-r-xr-xr-x   1 root    root       94529 Dec  6 19:07 ld-linux.so.2
-r--r--r--   1 root    root     1171196 Dec  6 19:07 libc.so.6
-r--r--r--   1 root    root      104744 Dec  6 19:07 libpthread.so.0
-r--r--r--   1 root    root       25596 Dec  6 19:07 librt.so.1

/home/ftp/pub:
total 2
drwxr-x-wx   2 root    root        1024 Dec  6 19:07 incoming
-rw-r--r--   1 root    root           6 Dec  6 19:11 test.bestand

/home/ftp/pub/incoming:
total 0
```

The files /home/ftp/etc/group and /home/ftp/etc/passwd are examples and must be replaced the real ones without real passwords:

```
# perl -ne 's/^([^:]+:)([^:]*)(:.*)$/\1*\3/; print;' /etc/passwd > etc/passwd
# perl -ne 's/^([^:]+:)([^:]*)(:.*)$/\1*\3/; print;' /etc/group > etc/group
```

The file welcome.msg contains the message that is displayed when the anonymous or ftp user successfully logs in. This is configured in /etc/wu-ftpd/ftpaccess as follows:

```
message /welcome.msg           login
```

The sequence of messages during a session is:

```
willem@pug:~$ ftp pug
Connected to pug.
220-########################################################################
220-THIS IS /etc/wu-ftpd/welcome.msg
220-configured with banner
```

```
220---
220-Welcome, archive user @pug !
220-
220-The local time is: Fri Dec  7 01:56:38 2001
220-
220-This is an experimental FTP server.  If have any unusual problems,
220-please report them via e-mail to <root@pug>.
220-
220-If you do have problems, please try using a dash (-) as the first
220-character of your password -- this will turn off the continuation
220-messages that may be confusing your FTP client.
220-##############################################################
220-
220-
220 configured with greeting: pug FTP Server at your service!
Name (pug:willem): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230-##############################################################
230-THIS IS /home/ftp/welcome.msg
230-configured with message
230---
230-Welcome, archive user anonymous@pug !
230-
230-The local time is: Fri Dec  7 01:57:05 2001
230-
230-This is an experimental FTP server.  If have any unusual problems,
230-please report them via e-mail to <root@pug>.
230-
230-If you do have problems, please try using a dash (-) as the first
230-character of your password -- this will turn off the continuation
230-messages that may be confusing your FTP client.
230-##############################################################
230-
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Welcome message for all ftp users

This is configured in the /etc/wu-ftpd/ftpaccess **in the following manner**:

```
banner /etc/wu-ftpd/welcome.msg
```

This message will be shown before the user logs in:

```
S ftp pug
Connected to pug.
220-##############################################################
220-THIS IS /etc/wu-ftpd/welcome.msg
```

```
220-
220-Welcome, archive user @pug !
220-
220-The local time is: Fri Dec  7 01:33:52 2001
220-
220-This is an experimental FTP server.  If have any unusual problems,
220-please report them via e-mail to <root@pug>.
220-
220-If you do have problems, please try using a dash (-) as the first
220-character of your password -- this will turn off the continuation
220-messages that may be confusing your FTP client.
220-##############################################################################
220-
220-
220 pug FTP server ready.
Name (pug:willem):
```

Login message for all not chrooted users

This is configured in the /etc/wu-ftpd/ftpaccess file in the following manner:

```
message /welcome.msg          login
```

The message for the chrooted users is in a file with the same name but must reside in the root directory of the chrooted user.

Directory specific messages

A directory message is a message that tells something about the contents of the directory. This message should be put in the file that you specify in /etc/wu-ftpd/ftpaccess. This is done in the following manner:

```
message .message              cwd=*
```

This means that a file with the name .message is displayed when users enter the directory containing the file.

Preventing all ftp connections

The wu-ftpd daemon does not care if the file /etc/nologin exists. Whether users can ftp or not is configured in the files /etc/wu-ftpd/ftpaccess and /etc/ftpusers.

Preventing specific users or groups from using ftp

This can be done by adding a username to the file /etc/ftpusers or by adding deny-uid and/or deny-gid lines to the file /etc/wu-ftpd/ftpaccess. Consult the man page for further details.

Restricting specific users to their home directories

With wu-ftpd, there are three types of users: anonymous, guest and real. Only real users can move around the filesystem according to their access privileges. Specific users can be restricted to their home directories by defining them as guest users. This is done in the file /etc/wu-ftpd/ftpaccess in the following manner:

guestgroup <groupname> [<groupname> ...]
guestuser <username> [<username> ...]

Consult the man page for further details.

Additional precautions

If you allow users to upload data into an incoming directory, do not allow the creation of sub-directories in that directory to prevent misuse.

Secure shell (OpenSSH) (2.212.4)

The candidate should be able to configure sshd to allow or deny root logins, enable or disable X forwarding. This objective includes generating server keys, generating a user's public/private key pair, adding a public key to a user's authorized_keys file and configuring ssh-agent for all users. Candidates should also be able to configure port-forwarding to tunnel an application protocol over ssh, configure ssh to support the ssh protocol versions 1 and 2, disable non-root logins during system maintenance, configure trusted clients for ssh logins without a password and make multiple connections from multiple hosts to guard against loss of connection to a remote host following configuration changes.

Key files, terms and utilities include:

ssh, sshd
/etc/ssh/sshd_config
~/.ssh/id_dsa.pub and id_sa, ~/.ssh/ authorized_keys
.shosts, .rhosts

What are ssh and sshd?

ssh is a client program for logging into a remote machine and for executing commands on a remote machine. It is intended to replace **rlogin** and **rsh**, and provide secure encrypted communications between two untrusted hosts over an insecure network.

sshd is the server (daemon) program for ssh. Together these programs replace **rlogin** and **rsh**, and provide secure encrypted communications between two untrusted hosts over an insecure network.

You can copy files using the **scp** command. Scp stands for Secure CoPy. **scp** uses **ssh** for data transfer.

Installing ssh and sshd

There are several ways to do this, I use the Debian way:

apt-get install ssh

To check which files are installed type this command:

dpkg --listfiles ssh

Configuring sshd

Configuring **sshd** is done by editing the configuration file /etc/ssh/sshd_config.

Allow or deny root logins

This is done by setting the keyword **PermitRootLogin** in the configuration file to the appropriate value. To make it more difficult for someone to gain full access, you shouldn't allow root logins.

Possible values for PermitRootLogin:

yes

: This is the default. When set to yes, root can login using **ssh**.

no

: When set to no, root cannot login using **ssh**.

without-password

: This means that password authentication is disabled for root.

forced-commands-only

: This means that root can only use **ssh** to login to the system and execute the commands that are given on the command line.

Allow or deny non-root logins

There are a number of keywords that can be used to influence the behaviour of **sshd** in relation to logins.

These keywords are:

AllowUsers

: This keyword is followed by a list of user names, separated by spaces. Login is allowed only for usernames that match one of the patterns. You can use "*" and "?" as wildcards in the patterns.

DenyUsers

This keyword is followed by a list of user names, separated by spaces. User names that match one of the patterns can not login. You can use "*" and "?" as wildcards in the patterns.

AllowGroups

This keyword is followed by a list of group names, separated by spaces. Login is allowed only for users who are a member of one or more groups that match one of the patterns. You can use "*" and "?" as wildcards in the patterns.

DenyGroups

This keyword is followed by a list of group names, separated by spaces. Login is not allowed for users who are a member of one or more groups that match one of the patterns. You can use "*" and "?" as wildcards in the patterns.

Enabling or disabling X forwarding

There are a number of keywords that can be used to influence the behaviour of **sshd** in relation to the X Windows system.

The keywords are:

X11Forwarding

X11 forwarding is a mechanism where the program runs on one machine and the X Windows output is shown on another machine. The command **ssh -X remote** will set the <variable>DISPLAY</variable> in the server-shell to **localhost:num:0** which is actually the tunnel-endpoint which is mapped back to the original <variable>DISPLAY</variable> in the client context. This tunnel is secured using ssh. X11Forwarding can be set to yes or no. The default is no.

X11DisplayOffset

This specifies the first display number that is available for the X11 forwarding of **sshd**. This prevents **sshd** from interfering with real servers. The default is 10.

XAuthLocation

This specifies the fully qualified location of the **xauth** command. The default location is /usr/bin/X11/xauth. **xauth** is used to edit and display the authorization information used in connecting to the X server.

If you connect to machine B from machine A using **ssh your_account@machineB** and start an **xterm** for instance, the process will run on machine B and the X output will go to machine A.

On machine A, you must authorise machine B for X with the command **xhost +machineB**.

Keys and their purpose

There are two types of keys: Server keys and User keys.

Server keys

There is a slight difference between the protocol versions 1 and 2.

The man page of sshd describes this as follows:

SSH protocol version 1

> Each host has a host-specific RSA key (normally 1024 bits) used to identify the host. Additionally, when the daemon starts, it generates a server RSA key (normally 768 bits). This key is normally regenerated every hour if it has been used and is never stored on disk.
>
> Whenever a client connects the daemon responds with its public host and server keys. The client compares the RSA host key against its own database to verify that it has not changed. The client then generates a 256 bit random number. It encrypts this random number using both the host key and the server key, and sends the encrypted number to the server. Both sides then use this random number as a session key which is used to encrypt all further communications in the session. The rest of the session is encrypted using a conventional cipher, currently Blowfish or 3DES, with 3DES being used by default. The client selects the encryption algorithm to use from those offered by the server.
>
> Next, the server and the client enter an authentication dialog. The client tries to authenticate itself using .rhosts authentication, .rhosts authentication combined with RSA host authentication, RSA challenge- response authentication or password based authentication.
>
> Rhosts authentication is normally disabled because it is fundamentally insecure, but can be enabled in the server configuration file if desired. System security is not improved unless **rshd**, **rlogind**, **rexecd** and **rexd** are disabled.

SSH protocol version 2

> Version 2 works similarly: Each host has a host-specific DSA key used to identify the host. However, when the daemon starts, it does not generate a server key. Forward security is provided through a Diffie-Hellman key agreement. This key agreement results in a shared session key.
>
> The rest of the session is encrypted using a symmetric cipher, currently 128-bit AES, Blowfish, 3DES, CAST128, Arcfour, 192-bit AES or 256-bit AES. The client selects the encryption algorithm to use from those offered by the server. Additionally, session integrity is provided through a cryptographic message authentication code (hmac-sha1

or hmac- md5).

Protocol version 2 provides a public key based user (PubkeyAuthentication) or client host (HostbasedAuthentication) authentication method, conventional password authentication and challenge-response based methods.

If possible, choose SSH protocol version 2 as the only possible or as the first protocol to use.

User keys, public and private

ssh implements the RSA authentication protocol automatically. The user creates an RSA key pair by running **ssh-keygen**. This stores the private key in $HOME/.ssh/id_dsa and the public key in $HOME/.ssh/id_dsa.pub in the user's home directory. The user should then copy the id_dsa.pub to $HOME/.ssh/authorized_keys in his home directory on the remote machine. The authorized_keys file has one key per line which can be very long. After this, the user can log in without giving the password. Instead of dsa, rsa can be used also. The names of the keys reflect this.

Configuring the ssh-agent

ssh-agent is a program to hold private keys used for public-key authentication (RSA, DSA). The idea is that **ssh-agent** is started in the beginning of an X-session or a login session, and all other windows or programs are started as clients to the **ssh-agent** program. Through use of environment variables, the agent can be located and automatically used for authentication when the logging-in to other machines using **ssh**.

Login session

Add the following two lines to your $HOME/.bash_profile file or equivalent (depending on the shell you are using) to be able to login without having to type your password each time:

```
eval ‘ssh-agent‘
ssh-add
```

The **eval `ssh-agent`** sets a number of environment variables. In fact, **ssh-agent** returns the strings needed to set them and **eval** sees to it that they get set.

The **ssh-add** command without parameters reads the contents of the file $HOME/.ssh/id_dsa which contains the private key, as described earlier.

When the user logs out from the local system, the program **ssh-agent** must be terminated. To see to it that this happens automatically, add the following line to your .bash_logout file or equivalent, depending on the shell you are using:

ssh-agent -k

The process id of of the current agent is determined by examining the contents of the environment variable $SSH_AGENT_PID, which has been set by the **eval `ssh-agent`** command.

To set this up for all future users, modify files in the /etc/skel directory which automatically get copied to the new user's home directory when this user is created with **adduser** or with the **useradd -m [-k skeleton directory]**.

Enabling X-sessions

There are several ways to do this, depending on how X is started and which display manager you are using.

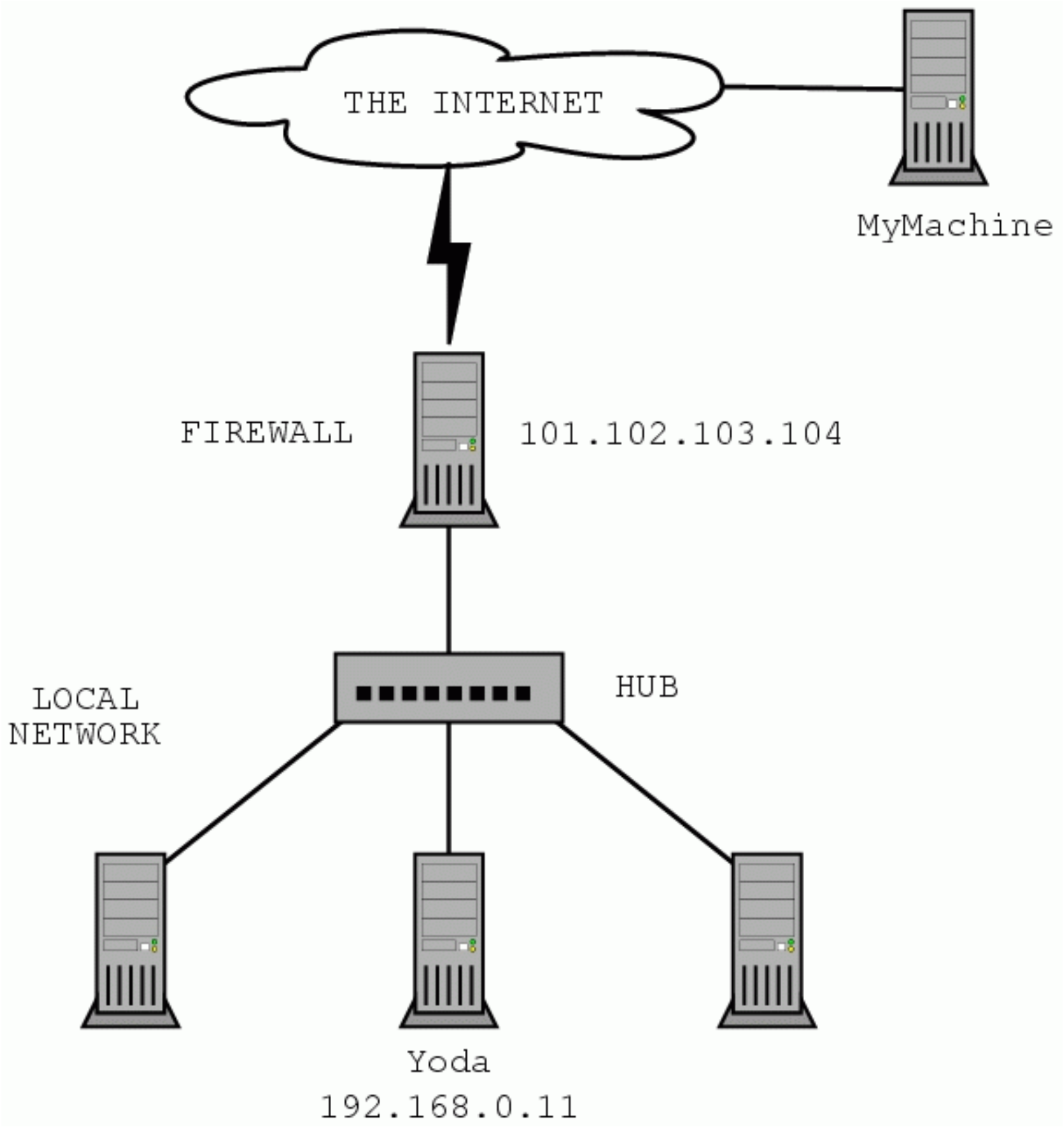
If you start X from the command line with **startx**, you can type **ssh-agent startx**, open a terminal window in X and type **ssh-add**, which will prompt you for the passphrase and load your keys.

Since I am using Gnome and Debian, I have installed the graphical version of **ssh-askpass** by installing the appropriate .deb package:

apt-get install ssh-askpass-gnome

This file is called **gnome-ssh-askpass** and is started by the **ssh-agent** command which, in its turn, is automatically started by Gnome. This is configured in the file /etc/gdm/Session/Gnome.

Tunneling an application protocol over ssh with portmapping



As an example, consider the situation shown in the picture. We are working on MyMachine, and we want to get the file /home/willem/fs0_backup from the machine called Yoda using **scp**.

Without ssh tunneling with port forwarding, we would have to do this in a few steps, assuming there is a user willem on both the firewall and on yoda:

```
on MyMachine do: ssh willem@101.102.103.104    'login on the firewall
on Firewall  do: scp willem@yoda:fs0_backup .  'place file from yoda on firewall
on MyMachine do: scp willem@101.102.103.104:fs0_backup .  'get the file
on MyMachine do: ssh willem@101.102.103.104 rm fs0_backup 'remove the file
```

With tunneling, this can be done with one command, not including the command to set up the tunnel itself.

Before we get to the actual commands involved, let me explain what tunneling with port forwarding means. It is quite simple actually. What happens is that we use **ssh** to tell MyMachine that it has got a port that points to port 22, the port that is used by **ssh**, on yoda. After that, all data sent to that port on MyMachine will get forwarded to port 22 on yoda.

Let us assume that the local port is going to be port 8022. The **ssh** command used to create the tunnel becomes:

```
MyMachine # ssh -L 8022:yoda:22 willem@101.102.103.104
```

We are now logged in on the Firewall as the user willem and the tunnel has been established. We can now open a second terminal window and type the **scp** command that makes use of that tunnel and gets the file:

```
MyMachine $ scp -P willem@localhost:fs0_backup .
```

And we've got the file! The tunnel can be terminated by logging out from the firewall.

We are not quite there though. This can also be done in a different way without the need for two windows and without having to terminate the tunnel manually.

This is accomplished by the same set of commands we used earlier but we run the tunnel in the background. To be able to run the tunnel in the background without this resulting in the immediate closure of that tunnel, we have to fool the tunnel to stay open long enough. This is done by using **ssh**'s ability to execute a remote command and terminate after the completion of that command, combined with the "-f" flag to tell **ssh** to run in the background. The command that we will execute on the other machine is **sleep 60** which gives us 60 seconds to initiate the **scp** command. Yes, to INITIATE the connection. It does not matter how long the actual **scp** command takes, the connection will not be terminated before the **scp** is finished.

To illustrate the above, we will combine both commands on one line and use **sleep 1** instead of **sleep 60**:

```
MyMachine $ ssh -f -L 8022:yoda:22 willem@101.102.103.104 sleep 1;\
```

```
scp -P8022 willem@localhost:fs0_backup .

willem@localhost's password: Waiting for forwarded connections to terminate...
The following connections are open:
  #0 direct-tcpip: listening port 8022 for yoda port 22, connect from
127.0.0.1 port 1475 (t4 r1 i1/0 o16/0)

<here we type the password>
fs0_backup          100% |*****************************|  6525       00:00
MyMachine $
```

As you can see, we immediately got a timeout because we were not quick enough in typing the password. What you can also see is that this did not end the connection. As soon as the password was given, the file was copied with the **scp** command.

Note that we had to tell **scp** with -P8022 to use port 8022 since it would otherwise have used the default port 22.

In this example we have used **scp** as an example but the same trick also works for other protocols, such as smtp.

Since the tunnel is an ssh tunnel, all data going through the tunnel is encrypted.

The .rhosts and .shosts files

The $HOME/.rhosts file was originally used by the **rlogin** and **rsh** commands. The file contained information about the machines the user can connect from. Say you create a $HOME/.shosts file which contains a line in the form:

```
machine-a.somedomain user-a machine-b.somedomain user-b
```

Then user-a can login to machine-b from machine-a using user-b's account without the need for user-b's password.

Although **ssh** can still make use of the $HOME/.rhosts and $HOME/.shosts files, this is configured in the /etc/ssh/sshd_config file and it is highly recommended you use the new authentication method. The new method uses the **ssh-keygen** command and the $HOME/.ssh/authorized_keys files, which are more secure.

Securing FTP servers (2.212.3) Home TCP_wrappers (2.212.5)

TCP_wrappers (2.212.5)

The candidate should be able to configure tcpwrappers to allow connections to specified servers from only certain hosts or subnets.

Key files, terms and utilities include:

inetd.conf,
tcpd
hosts.allow, hosts.deny
xinetd

What do tcp wrappers do?

Before tcp wrappers were born, a daemon called **inetd** was listening for incoming network connections and took it upon itself to start the appropriate server program when needed.

For security reasons, it became necessary to only allow certain type of incoming connections from certain hosts. The **inetd** daemon listens to incoming requests and instead of starting the server program needed, **inetd** starts **tcpd** which does some additional checks (such as where the request came from). If **tcpd** determines that the connection should be honored, the server program needed is launched by **tcpd**.

What don't tcp wrappers do?

Tcp wrappers do not protect against network sniffing because tcp wrappers do not encrypt the traffic. Use **ssh** encryption to prevent network sniffing.

Configuring tcp wrappers

The first thing to do when you want a service to be started by **tcpd** is to tell **inetd** that this is the case. To tell **inetd** that we want the ftp server **wu-ftpd** to be started by **tcpd**, we must add the following line to **inetd**'s configuration file /etc/inetd.conf:

```
ftp   stream  tcp nowait  root  /usr/sbin/tcpd  /usr/sbin/wu-ftpd -l
```

The lines of /etc/inetd.conf are made up of the following mandatory fields:

service name

This is the name of the service as specified in the file /etc/services, in this case "ftp".

socket type

This is the socket type which should be one of "stream", "dgram", "raw", "rdm" or "seqpacket". In this case the socket type for ftp is "stream".

protocol

This is the protocol type used as specified in the file /etc/protocols, in this case "tcp".

wait/nowait[.max]

For non datagram sockets this is always "nowait".

user[.group]

This entry specifies which user/group the program should run with. In this case, "root".

server program

This entry contains the pathname of the program to be executed by **inetd** when a request is made to that socket.

server program arguments

This is the program, with it's command line arguments, that is to be started by **inetd** if the criteria are met. In this case, this is **/usr/sbin/wu-ftpd -l**.

The next thing to do is to edit the contents of the files /etc/hosts.allow and /etc/hosts.deny which both use the same format:

daemon_list : client_list [: shell_command]

daemon_list

This is a list of one or more daemon process names or wildcards. Consult the man page of host_access(5) for details.

client_list

This is a list of one or more host names, host addresses, patterns or wildcards that will be matched against the client host name or address. Consult the man page of host_access(5) for details.

shell_command

Command to be run by the shell when the rule matches.

The access control software consults two files. The search stops at the first match: Access will be granted when a (daemon,client) pair matches an entry in the /etc/hosts.allow file.

Or access will be denied when a (daemon,client) pair matches an entry in the /etc/hosts.deny file.

Otherwise, access will be granted. A non-existing access control file is treated as an empty file. Thus, access control can be turned off by providing no access control files.

xinetd

This stands for "eXtended InterNET services Daemon" and replaces the combo **inetd** and **tcpd**. **xinetd** can do the same things and offers extra functionality such as: more sophisticated access control, preventing DoS attacks, extensive logging. For more information, take a look at the *Xinetd Homepage* .

Security tasks (2.212.6)

The candidate should be able to install and configure kerberos and perform basic security auditing of source code. This objective includes subscribing to security alerts from Bugtraq, CERT, CIAC or other sources, being able to test for open mail relays and anonymous FTP servers and installing and configuring an intrusion detection system, such as snort or Tripwire. Candidates should also be able to update the IDS configuration as new vulnerabilities are discovered and apply security patches and bugfixes.

Key files, terms and utilities include:

telnet
Tripwire
nmap

Kerberos

What is Kerberos?

Kerberos was created by the *Massachusetts Institute of Technology*. All information in this section comes from their site.

Kerberos is a network-authentication protocol for client/server applications. Kerberos supports strong authentication and data-encryption. This makes it possible for both the client and the server to assure themselves about their respective identities and, once they are satisfied, use encrypted to communication to keep transmitted passwords and data private.

Under Kerberos, a client (generally either a user or a service) sends a request for a ticket to the Key Distribution Center (KDC). The KDC creates a ticket-granting ticket (TGT) for the client, encrypts it using the client's password as the key and sends the encrypted TGT back to the client. The client then attempts to decrypt the TGT, using its password. If the client successfully decrypts the TGT (i. e., if the client gave the correct password), it keeps the decrypted TGT, which indicates proof of the client's identity.

The TGT, which expires after a specified time, permits the client to obtain additional tickets, which give permission for specific services. The requesting and granting of these additional tickets is user-transparent.

Preparing the installation

Before installing Kerberos V5, it is necessary to consider the following issues:

The name of your Kerberos realm (or the name of each realm, if you need more than one).

How you will map your hostnames onto Kerberos realms.

Which ports your KDC and kadmin (database access) services will use.

How many slave KDCs you need and where they should be located.

The hostnames of your master and slave KDCs.

How frequently you will propagate the database from the master KDC to the slave KDCs.

Whether you need backward compatibility with Kerberos V4.

Kerberos Realms

Although your Kerberos realm can be any ASCII string, convention is to make it the same as your domain name, in upper-case letters. For example, hosts in the domain fubar.org would be in the Kerberos realm FUBAR.ORG.

If you need multiple Kerberos realms, MIT recommends that you use descriptive names which end with your domain name, such as BOSTON.FUBAR.ORG and HOUSTON.FUBAR.ORG.

Mapping Hostnames onto Kerberos Realms

Mapping hostnames onto Kerberos realms is done in one of two ways.

The first mechanism, which has been in use for years in MIT-based Kerberos distributions, works through a set of rules in the krb5.conf configuration file. (See section krb5.conf.) You can specify mappings for an entire domain or subdomain and/or on a hostname-by-hostname basis. Since greater specificity takes precedence, you do this by specifying the mappings for a given domain or subdomain and listing the exceptions.

The Kerberos V5 System Administrator's Guide contains a thorough description of the parts of the krb5.conf file and what may be specified in each. A sample krb5.conf file appears in section krb5. conf. You should be able to use this file, substituting the relevant information for your Kerberos installation for the samples.

The second mechanism, recently introduced into the MIT code base but not currently used by default, works by looking up the information in special TXT records in the Domain Name Service. If this mechanism is enabled on the client, it will try to look up a TXT record for the DNS name formed by putting the prefix _kerberos in front of the hostname in question. If that record is not found, it will try using _kerberos and the host's domain name, then its parent domain, and so forth. So, for the hostname BOSTON.ENGINEERING.FOOBAR.COM, the names looked up would be:

_kerberos.boston.engineering.foobar.com
_kerberos.engineering.foobar.com
_kerberos.foobar.com
_kerberos.com

The value of the first TXT record found is taken as the realm name.

> **Note**
>
> Obviously, this doesn't work all that well if a host and a subdomain have the same name, and different realms. If, for example all the hosts in the ENGINEERING.FOOBAR.COM domain are in the ENGINEERING.FOOBAR.COM realm, but a host named ENGINEERING.FOOBAR.COM is for some reason in another realm. In that case, you would set up TXT records for all hosts, rather than relying on the fallback to the domain name.

Even if you do not choose to use this mechanism within your site, you may wish to set it up anyway, for use when interacting with other sites.

Ports for the KDC and Admin Services

The default ports used by Kerberos are port 88 for the KDC and port 749 for the admin server. You can, however, choose to run on other ports, as long as they are specified in each host's /etc/services and krb5.conf files, and the kdc.conf file on each KDC.

For a more thorough treatment of port numbers used by the Kerberos V5 programs, refer to the "Configuring Your Firewall to Work With Kerberos V5" section of the Kerberos V5 System Administrator's Guide, which is also available from the *Massachusetts Institute of Technology* .

Slave KDCs

Slave KDCs provide an additional source of Kerberos ticket-granting services in the event of inaccessibility of the master KDC. The number of slave KDCs you need and the decision of where to place them, both physically and logically, depends on the specifics of your network.

Kerberos authentication on your network requires that each client be able to contact a KDC. Therefore, you need to anticipate that a KDC might be unavailable and have a slave KDC to take up the slack.

Some considerations include:

Have at least one slave KDC as a backup for when the master KDC is down, is being upgraded or is otherwise unavailable.

If your network is split such that a network outage is likely to cause a network partition (some segment or segments of the network become cut off or isolated from other segments), have a slave KDC accessible to each segment.

If possible, have at least one slave KDC in a different building from the master, in case of power outages, fires or other localized disasters.

Hostnames for the Master and Slave KDCs

MIT recommends that your KDCs have a predefined set of CNAME records (DNS hostname aliases),

such as kerberos for the master KDC and kerberos-1, kerberos-2, ... for the slave KDCs. This way, if you need to swap a machine, you only need to change a DNS entry, rather than having to change hostnames.

A new mechanism for locating KDCs of a realm through DNS has been added to the MIT Kerberos V5 distribution. A relatively new record type called SRV has been added to DNS. Looked up by a service name and a domain name, these records indicate the hostname and port number to contact for that service, optionally with weighting and prioritizing. (See RFC 2782 if you want more information. You can follow the example below for straightforward cases.)

The use with Kerberos is fairly straightforward. The domain name used in the SRV record name is the domain-style Kerberos realm name. (It is possible to have Kerberos realm names that are not DNS-style names, but we don't recommend it for Internet use, and the code does not support it well.)

Several different Kerberos-related service names are used:

_kerberos._udp

This is for contacting any KDC. This entry will be used the most often. Normally you should list ports 88 and 750 on each of your KDCs.

_kerberos-master._udp

This entry should refer to those KDCs, if any, that will immediately see password changes to the Kerberos database. This entry is used for only one case: when a user is logging in and the password appears to be incorrect. The master KDC is then contacted (in case the user's password has recently been changed and the slave KDC hasn't been updated yet) and the password is tried again. Only if that fails an "incorrect password" error is given. If you have only one KDC, or for whatever reason there is no accessible KDC that would get database changes faster than the others, you do not need to define this entry.

_kerberos-adm._tcp

This should list port 749 on your master KDC. Support for it is not complete at this time, but it will eventually be used by the kadmin program and related utilities. For now, you will also need the admin_server entry in krb5.conf.

_kpasswd._udp

This should list port 464 on your master KDC. It is used when a user changes her password.

Be aware, however, that the DNS SRV specification requires that the hostnames listed be the canonical names, not aliases. So, for example, you might include the following records in your (BIND-style) zone file:

```
$ORIGIN foobar.com.
_kerberos          TXT     "FOOBAR.COM"
kerberos           CNAME   daisy
kerberos-1         CNAME   use-the-force-luke
kerberos-2         CNAME   bunny-rabbit
_kerberos._udp     SRV     0 0 88 daisy
```

```
                    SRV     0 0 88 use-the-force-luke
                    SRV     0 0 88 bunny-rabbit
_kerberos-master._udp   SRV     0 0 88 daisy
_kerberos-adm._tcp      SRV     0 0 749 daisy
_kpasswd._udp           SRV     0 0 464 daisy
```

As with the DNS-based mechanism for determining the Kerberos realm of a host, we recommend distributing the information this way for use by other sites that may want to interact with yours using Kerberos, even if you don't immediately make use of it within your own site. If you anticipate installing a very large number of machines on which it will be hard to update the Kerberos configuration files, you may wish to do all of your Kerberos service lookups via DNS and not put the information (except for admin_server, as noted above) in future versions of your krb5.conf files at all. Eventually, MIT hopes to phase out the listing of server hostnames in the client-side configuration files; making preparations now will make the transition easier in the future.

Database Propagation

The Kerberos database resides on the master KDC and must be propagated regularly (usually by a cron job) to the slave KDCs. In deciding how frequently the propagation should happen, you will need to balance the amount of time the propagation takes against the maximum reasonable amount of time a user should have to wait for a password change to take effect.

If the propagation time is longer than this maximum reasonable time (e.g., you have a particularly large database, you have a lot of slaves, or you experience frequent network delays), you may wish to cut down on your propagation delay by performing the propagation in parallel. To do this, have the master KDC propagate the database to one set of slaves, and then have each of these slaves propagate the database to additional slaves.

Installation and Configuration

Installing KDCs

The Key Distribution Centers (KDCs) issue Kerberos tickets. Each KDC contains a copy of the Kerberos database. The master KDC contains the master copy of the database, which it propagates to the slave KDCs at regular intervals. All database changes (such as password changes) are made on the master KDC.

Slave KDCs provide Kerberos ticket-granting services, but not database administration. This allows clients to continue to obtain tickets when the master KDC is unavailable.

MIT recommends that you install all of your KDCs to be able to function as either the master or one of the slaves. This will enable you to easily switch your master KDC with one of the slaves if necessary. (See section Switching Master and Slave KDCs.) This installation procedure is based on that recommendation. Install the Master KDC.

This installation procedure will require you to go back and forth a couple of times between the master KDC and each of the slave KDCs.

The first few steps must be done on the master KDC.

Edit the Configuration Files

Modify the configuration files, /etc/krb5.conf (see section krb5.conf) and /usr/local/var/krb5kdc/kdc.conf (see section kdc.conf) to reflect the correct information (such as the hostnames and realm name) for your realm. MIT recommends that you keep krb5.conf in /etc.

Among the settings in your /etc/krb5.conf file, be sure to create a logging stanza so that the **KDC** and **kadmind** will generate logging output. For example:

```
[logging]
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmin.log
    default = FILE:/var/log/krb5lib.log
```

Create the Database

You will use the **kdb5_util** command on the Master KDC to create the Kerberos database and the optional stash file. The stash file is a local copy of the master key that resides in encrypted form on KDC's local disk. The stash file is used to authenticate the KDC to itself automatically before starting the **kadmind** and **krb5kdc** daemons (e.g., as part of the machine's boot sequence). The stash file, like the keytab file (see See section The Keytab File, for more information) is a potential point-of-entry for a break-in, and if compromised, would allow unrestricted access to the Kerberos database. If you choose to install a stash file, it should be readable only by root and should exist only on KDC's local disk. The file should not be part of any backup of the machine, unless access to the backup data is secured as tightly as access to the master password itself.

Note that **kdb5_util** will prompt you for the master key for the Kerberos database. This key can be any string. A good key is one you can remember, but that no one else can guess. Examples of bad keys are words that can be found in a dictionary, any common or popular name, especially a famous person (or cartoon character), your username in any form (e.g., forward, backward, repeated twice, etc.), and any of the sample keys that appear in this document. One example of a key which might be good if it did not appear in this document is "MITiys4K5!", which represents the sentence "MIT is your source for Kerberos 5!" (It's the first letter of each word, substituting the numeral "4" for the word "for", and includes the punctuation mark at the end.)

The following is an example of how to create a Kerberos database and stash file on the master KDC, using the **kdb5_util** command. Replace ATHENA.MIT.EDU with the name of your Kerberos realm.

```
shell% /usr/local/sbin/kdb5_util create -r ATHENA.MIT.EDU -s
Initializing database '/usr/local/var/krb5kdc/principal' for          \
realm 'ATHENA.MIT.EDU',
master key name 'K/M@ATHENA.MIT.EDU'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key: { Type the master password }
Re-enter KDC database master key to verify: { Type it again }
```

shell%

This will create five files in the directory specified in your kdc.conf file: two Kerberos database files, principal.dband principal.ok; the Kerberos administrative database file, principal.kadm5; the administrative database lock file, principal.kadm5.lock and the stash file, .k5stash. (The default directory is /usr/local/var/krb5kdc.) If you do not want a stash file, run the above command without the -s option.

Add Administrators to the Acl File

Next, you need to create an Access Control List (acl) file and put the Kerberos principal of at least one of the administrators into it. The filename should match the value you have set for "acl_file" in your kdc.conf file. The default file name is kadm5.acl. The format of the file is:

Kerberos principal permissions optional target principal

The Kerberos principal (and optional target principal) can include the "*" wildcard, so if you want any principal with the instance "admin" to have full permissions on the database, you could use the principal "*/admin@REALM" where "REALM" is your Kerberos realm.

Note: a common use of an admin instance is so you can grant separate permissions (such as administrator access to the Kerberos database) to a separate Kerberos principal. For example, the user joeadmin might have a principal for his administrative use, called joeadmin/admin. This way, joeadmin would obtain joeadmin/admin tickets only when he actually needs to use those permissions. Refer to the Kerberos V5 Administrator's Guide or the Kerberos V5 User's Guide for more detailed explanations of principals and instances.

The permissions (acls) recognized in the acl file are the following:

a

allows the addition of principals or policies in the database.

A

prohibits the addition of principals or policies in the database.

d

allows the deletion of principals or policies in the database.

D

prohibits the deletion of principals or policies in the database.

m

allows the modification of principals or policies in the database.

M

prohibits the modification of principals or policies in the database.

c

allows the changing of passwords for principals in the database.

C

prohibits the changing of passwords for principals in the database.

i

allows inquiries to the database.

I

prohibits inquiries to the database.

l

allows the listing of principals or policies in the database.

L

prohibits the listing of principals or policies in the database.

*

Short for all privileges (admcil).

x

Short for all privileges (admcil); identical to "*".

To give the principal */admin@ATHENA.MIT.EDU permission to change all of the database permissions on any principal permissions, you would place the following line in the file:

```
*/admin@ATHENA.MIT.EDU  *
```

To give the principal joeadmin@ATHENA.MIT.EDU permission to add, list, and inquire about any principal that has the instance "root", you would add the following line to the acl file:

```
joeadmin@ATHENA.MIT.EDU  ali  */root@ATHENA.MIT.EDU
```

Add Administrators to the Kerberos Database

Next you need to add administrative principals to the Kerberos database. (You must add at least one now.) To do this, use **kadmin.local** on the master KDC. The administrative principals you create should be the ones you added to the ACL file. (See the section Add Administrators to the Acl File.) In the following example, the administration principal admin/admin is created:

```
shell% /usr/local/sbin/kadmin.local
kadmin.local: addprinc admin/admin@ATHENA.MIT.EDU
```

```
WARNING: no policy specified for "admin/admin@ATHENA.MIT.EDU";
defaulting to no policy.
Enter password for principal admin/admin@ATHENA.MIT.EDU: { Enter a password }
Re-enter password for principal admin/admin@ATHENA.MIT.EDU: { Type it again }
Principal "admin/admin@ATHENA.MIT.EDU" created.
kadmin.local:
```

Create a kadmind Keytab

The kadmind keytab is the key that kadmind will use to decrypt administrators' Kerberos tickets to determine whether or not it should give them access to the database. You need to create the kadmin keytab with entries for the principals kadmin/admin and kadmin/changepw. (These principals are placed in the Kerberos database automatically when you create it.) To create the kadmin keytab, run **kadmin.local** and use the **ktadd** command, as in the following example.

```
shell% /usr/local/sbin/kadmin.local
kadmin.local: ktadd -k /usr/local/var/krb5kdc/kadm5.keytab              \
kadmin/admin kadmin/changepw
Entry for principal kadmin/admin@ATHENA.MIT.EDU with
    kvno 3, encryption type DES-CBC-CRC added to keytab
    WRFILE:/usr/local/var/krb5kdc/kadm5.keytab.
Entry for principal kadmin/changepw@ATHENA.MIT.EDU with
    kvno 3, encryption type DES-CBC-CRC added to keytab
    WRFILE:/usr/local/var/krb5kdc/kadm5.keytab.
kadmin.local: quit
shell%
```

As specified in the "-k" argument, ktadd will save the extracted keytab as /usr/local/var/krb5kdc/kadm5.keytab. The filename you use must be the one specified in your kdc.conf file.

Start the Kerberos Daemons on the Master KDC

At this point, you are ready to start the Kerberos daemons on the Master KDC. To do so, type:

```
shell% /usr/local/sbin/krb5kdc
shell% /usr/local/sbin/kadmind
```

Each daemon will fork and run in the background. Assuming you want these daemons to start up automatically at boot time, you can add them to KDC's /etc/rc or /etc/inittab file. You need to have a stash file in order to do this.

You can verify that they started properly by checking for their startup messages in the logging locations you defined in /etc/krb5.conf. (See the section called " Edit the Configuration Files ") For example:

```
shell% tail /var/log/krb5kdc.log
Dec 02 12:35:47 beeblebrox krb5kdc[3187](info): commencing operation
shell% tail /var/log/kadmin.log
Dec 02 12:35:52 beeblebrox kadmind[3189](info): starting
```

Any errors the daemons encounter while starting will also be listed in the logging output.

Install the Slave KDCs

You are now ready to start configuring the slave KDCs. Assuming you are setting the KDCs up so that you can easily switch the master KDC with one of the slaves, you should perform each of these steps on the master KDC as well as the slave KDCs, unless these instructions specify otherwise.

Create Host Keys for the Slave KDCs

Each KDC needs a host principal in the Kerberos database. You can enter these from any host, once the kadmind daemon is running. If, for example your master KDC were called kerberos.mit.edu, and you had two KDC slaves named kerberos-1.mit.edu and kerberos-2.mit.edu, you would type the following:

```
shell% /usr/local/sbin/kadmin
kadmin: addprinc -randkey host/kerberos.mit.edu
WARNING: no policy specified for "host/kerberos.mit.edu@ATHENA.MIT.EDU";
defaulting to no policy.
Principal "host/kerberos.mit.edu@ATHENA.MIT.EDU" created.
kadmin: addprinc -randkey host/kerberos-1.mit.edu
WARNING: no policy specified for "host/kerberos-1.mit.edu@ATHENA.MIT.EDU";
defaulting to no policy.
Principal "host/kerberos-1.mit.edu@ATHENA.MIT.EDU" created.
kadmin: addprinc -randkey host/kerberos-2.mit.edu
WARNING: no policy specified for "host/kerberos-2.mit.edu@ATHENA.MIT.EDU";
defaulting to no policy.
Principal "host/kerberos-2.mit.edu@ATHENA.MIT.EDU" created.
kadmin:
```

It is not actually necessary to have the master KDC server in the Kerberos database, but it can be handy if anyone will be logging into the machine as something other than root, or you want to be able to swap the master KDC with one of the slaves if necessary.

Extract Host Keytabs for the KDCs

Each KDC (including the master) needs a keytab to decrypt tickets. Ideally, you should extract each keytab locally on its own KDC. If this is not feasible, you should use an encrypted session to send them across the network. To extract a keytab on a KDC called kerberos.mit.edu, you would execute the following command:

```
kadmin: ktadd host/kerberos.mit.edu
kadmin: Entry for principal host/kerberos.mit.edu@ATHENA.MIT.EDU with
    kvno 1, encryption type DES-CBC-CRC added to keytab
    WRFILE:/etc/krb5.keytab.
kadmin:
```

Note that the principal must exist in the Kerberos database in order to extract the keytab.

Set Up the Slave KDCs for Database Propagation

The database is propagated from the master KDC to the slave KDCs via the **kpropd** daemon. To set up propagation, create a file on each KDC named /usr/local/var/krb5kdc/kpropd.acl containing the principals for each of the KDCs. If, for example the master KDC were kerberos.mit.edu, the slave KDCs were kerberos-1.mit.edu and kerberos-2.mit.edu, and the realm were ATHENA.MIT.EDU, then the file's contents would be:

```
host/kerberos.mit.edu@ATHENA.MIT.EDU
host/kerberos-1.mit.edu@ATHENA.MIT.EDU
host/kerberos-2.mit.edu@ATHENA.MIT.EDU
```

Then, add the following lines to /etc/inetd.conf file on each KDC.

```
krb5_prop stream tcp nowait root /usr/local/sbin/kpropd kpropd
eklogin   stream tcp nowait root /usr/local/sbin/klogind              \
klogind -k -c -e
```

The first line sets up the kpropd database propagation daemon. The second line sets up the eklogin daemon, allowing Kerberos-authenticated, encrypted rlogin to the KDC.

You also need to add the following lines to /etc/services on each KDC:

```
kerberos       88/udp     kdc      # Kerberos authentication (udp)
kerberos       88/tcp     kdc      # Kerberos authentication (tcp)
krb5_prop      754/tcp             # Kerberos slave propagation
kerberos-adm   749/tcp             # Kerberos 5 admin/changepw (tcp)
kerberos-adm   749/udp             # Kerberos 5 admin/changepw (udp)
eklogin        2105/tcp            # Kerberos encrypted rlogin
```

Back on the Master KDC

Now that the slave KDCs are able to accept database propagation, you'll need to propagate the database to each of them.

Propagate the Database to Each Slave KDC

First, create a dump of the database on the master KDC, as follows:

```
shell% /usr/local/sbin/kdb5_util dump /usr/local/var/krb5kdc/slave_datatrans
shell%
```

Next, you need to manually propagate the database to each slave KDC, as in the following example.

```
/usr/local/sbin/kprop -f /usr/local/var/krb5kdc/slave_datatrans        \
kerberos-1.mit.edu
/usr/local/sbin/kprop -f /usr/local/var/krb5kdc/slave_datatrans        \
kerberos-2.mit.edu
```

You will need a script to dump and propagate the database. The following is an example of a bourne shell script that will do this. Remember that you need to replace /usr/local with the name of the directory in which you installed Kerberos V5.

```
#!/bin/sh

kdclist = "kerberos-1.mit.edu kerberos-2.mit.edu"

/usr/local/sbin/kdb5_util -R "dump /usr/local/var/krb5kdc/slave_datatrans"

for kdc in $kdclist
do
/usr/local/sbin/kprop -f /usr/local/var/krb5kdc/slave_datatrans $kdc
done
```

You will need to set up a cron job to run this script at the intervals you decided on earlier (See section Database Propagation.)

Finish Installing the Slave KDCs

Now that the slave KDCs have copies of the Kerberos database, you can create stash files for them and start the krb5kdc daemon.

Create Stash Files on the Slave KDCs

Create stash files by issuing the following commands on each slave KDC:

```
shell% kdb5_util stash
kdb5_util: Cannot find/read stored master key while reading master key
kdb5_util: Warning: proceeding without master key
Enter KDC database master key: { Enter the database master key }
```

```
shell%
```

As mentioned above, the stash file is necessary for your KDCs to be able to authenticate themselves, such as when they reboot. You could run your KDCs without stash files, but you would then need to type in the Kerberos database master key by hand every time you start a KDC daemon.

Start the krb5kdc Daemon on Each KDC

The final step in configuring your slave KDCs is to run the KDC daemon:

```
shell% /usr/local/sbin/krb5kdc
```

As with the master KDC, you will probably want to add this command to the KDC's /etc/rc or /etc/inittab files, so they will start the krb5kdc daemon automatically at boot time.

Add Kerberos Principals to the Database

Once your KDCs are set up and running, you are ready to use **kadmin** to load principals for your users, hosts and other services into the Kerberos database. This procedure is described fully in the "Adding or Modifying Principals" section of the Kerberos V5 System Administrator's Guide. (See the section called " Create Host Keys for the Slave KDCs " for a brief description.) The keytab is generated by running **kadmin** and issuing the **ktadd** command.

Limit Access to the KDCs

To limit the possibility that your Kerberos database could be compromised, MIT recommends that each KDC be a dedicated host, with limited access. If your KDC is also a file server, FTP server, Web server, or even just a client machine, someone who obtained root access through a security hole in any of those areas could gain access to the Kerberos database.

MIT recommends that your KDCs use the following /etc/inetd.conf file.

```
#
# Time service is used for clock synchronization.
#
time    stream  tcp     nowait  root    internal
time    dgram   udp     wait    root    internal
#
# Limited Kerberos services
#
krb5_prop stream tcp nowait root /usr/local/sbin/kpropd  kpropd
eklogin   stream tcp nowait root /usr/local/sbin/klogind              \
klogind -5 -c -e
```

Switching Master and Slave KDCs

You may occasionally want to use one of your slave KDCs as the master. This might happen if you are upgrading the master KDC, or if your master KDC has a disk crash.

Assuming you have configured all of your KDCs to be able to function as either the master KDC or a slave KDC (as this document recommends), all you need to do to make the changeover is:

If the master KDC is still running, do the following on the old master KDC:

1. Kill the kadmind process.

2. Disable the cron job that propagates the database.

3. Run your database propagation script manually, to ensure that the slaves all have the latest copy of the database (see the section called " Propagate the Database to Each Slave KDC "). As of the 1.2.2 release, it is no longer necessary to use **kdb5_util dump-ov** in order to preserve per-principal policy information, as the default dump format now supports it. Note you should update your slaves prior to your master, so that they will understand the new dump format. (This is a good policy anyway.)

On the new master KDC:

1. Create a database keytab. (See section Create a kadmind Keytab.

2. Start the kadmind daemon. (See section Start the Kerberos Daemons on the Master KDC.)

3. Set up the cron job to propagate the database. (See section Propagate the Database to Each Slave KDC.)

4. Switch the CNAMEs of the old and new master KDCs. (If you don't do this, you'll need to change the krb5.conf file on every client machine in your Kerberos realm.)

Snort

What is it?

Snort is a lightweight network intrusion detection system capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/ matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts and much more. Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plugin architecture. Snort has a real-time alerting capability as well, incorporating alerting mechanisms for syslog, a user-specified file, a UNIX socket or WinPopup messages to Windows clients using Samba's smbclient.

Snort has three primary uses. It can be used as a straight packet-sniffer like **tcpdump**, a packet-logger (useful for network traffic debugging, etc), or as a full blown network-intrusion detection

system.

Snort logs packets in either **tcpdump** binary format or in Snort's decoded ASCII format to logging directories that are named based on the IP address of the "foreign" host.

Installation

You can get snort from *The Snort Download Center*. Since I am using Debian, I used the Debian way to install Snort:

```
# apt-get install snort
```

You will then be asked to specify the interface Snort should listen on. In my case, this is eth0.

Next you will be asked to specify how many IP addresses Snort should listen to. This is done in CIDR form, i.e. 192.168.2.0/24 for a block of 256 IP addresses or 192.168.2.8/32 for just one. I have chosen to only listen to my own IP address, which is 192.168.2.8.

Next you will be asked who should receive the daily statistic mails. The default is root.

To run snort in the sniffer mode, i.e. let snort dump TCP/IP packets to the screen type:

```
# snort -v
01/07-16:22:12.746540 192.168.2.8:1024 -> 192.168.2.1:53
UDP TTL:64 TOS:0x0 ID:13581 IpLen:20 DgmLen:60 DF
Len: 40
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

01/07-16:22:12.750346 192.168.2.1:53 -> 192.168.2.8:1024
UDP TTL:64 TOS:0x0 ID:58826 IpLen:20 DgmLen:422
Len: 402
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

01/07-16:22:12.751291 192.168.2.8 -> 198.186.203.20
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:25604   Seq:0  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

01/07-16:22:12.940559 198.186.203.20 -> 192.168.2.8
ICMP TTL:235 TOS:0x0 ID:11684 IpLen:20 DgmLen:84
Type:0  Code:0  ID:25604  Seq:0  ECHO REPLY
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The output shown above is the result of the command **ping www.debian.org**. The first section shows the outgoing DNS lookup for www.debian.org. The second section shows the reply to the DNS

lookup. The third section shows the outgoing ping (which is, in fact, an ICMP message of type ECHO). The fourth section shows the answer to the **ping** (which is, in fact, an ICMP message of type ECHO REPLY).

To run snort in the sniffer mode and show application data as well, type:

```
# snort -vd
01/07-16:21:17.979586 192.168.2.8:1024 -> 192.168.2.1:53
UDP TTL:64 TOS:0x0 ID:8105 IpLen:20 DgmLen:60 DF
Len: 40
57 35 01 00 00 01 00 00 00 00 00 00 03 77 77 77  W5...........www
06 64 65 62 69 61 6E 03 6F 72 67 00 00 01 00 01  .debian.org.....

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
01/07-16:21:17.983218 192.168.2.1:53 -> 192.168.2.8:1024
UDP TTL:64 TOS:0x0 ID:58825 IpLen:20 DgmLen:422
Len: 402
57 35 81 80 00 01 00 01 00 09 00 09 03 77 77 77  W5...........www
06 64 65 62 69 61 6E 03 6F 72 67 00 00 01 00 01  .debian.org.....
C0 0C 00 01 00 01 00 00 04 17 00 04 C6 BA CB 14  ................
C0 10 00 02 00 01 00 00 04 17 00 09 06 73 61 6D  .............sam
6F 73 61 C0 10 C0 10 00 02 00 01 00 00 04 17 00  osa.............
08 05 73 61 65 6E 73 C0 10 C0 10 00 02 00 01 00  ..saens.........
00 04 17 00 0A 07 6B 6C 65 63 6B 65 72 C0 10 C0  ......klecker...
10 00 02 00 01 00 00 04 17 00 0A 07 70 61 6E 64  ............pand
6F 72 61 C0 10 C0 10 00 02 00 01 00 00 04 17 00  ora.............
09 06 6D 75 72 70 68 79 C0 10 C0 10 00 02 00 01  ..murphy........
00 00 04 17 00 08 05 61 75 72 69 63 C0 10 C0 10  .......auric....
00 02 00 01 00 00 04 17 00 10 03 6E 73 32 07 63  ...........ns2.c
69 73 74 72 6F 6E 02 6E 6C 00 C0 10 00 02 00 01  istron.nl.......
00 00 04 17 00 0E 02 6E 73 05 68 61 6E 64 73 03  .......ns.hands.
63 6F 6D 00 C0 10 00 02 00 01 00 00 04 17 00 0A  com.............
03 6E 73 31 03 77 61 77 C0 DF C0 3C 00 01 00 01  .ns1.waw...<....
00 00 04 17 00 04 D1 F9 61 EA C0 51 00 01 00 01  ........a..Q....
00 02 3F A7 00 04 80 65 24 C0 C0 65 00 01 00 01  ..?....e$..e....
00 00 04 17 00 04 C6 BA CB 14 C0 7B 00 01 00 01  ...........{....
00 00 04 17 00 04 84 E5 89 F9 C0 91 00 01 00 01  ................
00 00 04 17 00 04 D8 EA E7 06 C0 A6 00 01 00 01  ................
00 00 04 17 00 04 CE F6 E2 2D C0 BA 00 01 00 01  .........-......
00 01 5E 34 00 04 C3 40 44 1C C0 D6 00 01 00 01  ..^4...@D.......
00 00 32 A0 00 04 C3 E0 35 27 C0 F0 00 01 00 01  ..2.....5'......
00 02 9D 5C 00 04 D5 29 79 FA                    ...\...)y.

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
01/07-16:21:17.984191 192.168.2.8 -> 198.186.203.20
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:24836   Seq:0  ECHO
```

```
3C 39 BC ED 00 0F 04 6A 08 09 0A 0B 0C 0D 0E 0F  <9.....j........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

01/07-16:21:18.178454 198.186.203.20 -> 192.168.2.8
ICMP TTL:235 TOS:0x0 ID:665 IpLen:20 DgmLen:84
Type:0  Code:0  ID:24836  Seq:0  ECHO REPLY
3C 39 BC ED 00 0F 04 6A 08 09 0A 0B 0C 0D 0E 0F  <9.....j........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The output shown above is the result of the command **ping www.debian.org**. The first section shows the outgoing DNS lookup for www.debian.org. The second section shows the reply to the DNS lookup. The third section shows the outgoing ping (which is, in fact, an ICMP message of type ECHO). The fourth section shows the answer to the **ping** (which is, in fact, an ICMP message of type ECHO REPLY).

To run snort in the packet-logger mode, creating directories based on the host, type:

```
# snort -dev -l ./snortlog
```

After starting snort and running the commands **ping www.debian.org** and **ping www.iaf.nl** the contents of the ./snortlog directory is as follows:

```
# ls -lR snortlog/
snortlog/:
total 4
drwx------   2 root    root       1024 Jan  8 09:56 192.168.2.8
drwx------   2 root    root       1024 Jan  8 09:56 198.186.203.20
drwx------   2 root    root       1024 Jan  8 09:56 80.89.228.10
-rw-------   1 root    root        528 Jan  8 09:56 ARP

snortlog/192.168.2.8:
total 8
-rw-------   1 root    root       3698 Jan  8 09:56 ICMP_ECHO
-rw-------   1 root    root       3295 Jan  8 09:56 UDP:1024-53

snortlog/198.186.203.20:
total 2
```

```
-rw-------   1 root     root       1609 Jan  8 09:56 ICMP_ECHO_REPLY

snortlog/80.89.228.10:
total 3
-rw-------   1 root     root       2138 Jan  8 09:56 ICMP_ECHO_REPLY
```

To run snort in network-intrusion detection mode, type:

```
# snort -d -h 192.168.2.0/24 -l ./snortlog -c snort.con
```

This lets snort run in the most basic network-intrusion detection mode, which logs the same way as the packet-logger mode.

There are quite a few of other possibilities, such as logging in a binary format, which can later be parsed by snort. Please consult the *Snort Users Manual* for more information.

Configuration

You can also create a configuration containing rules. To have Snort use these rules, use the "-c <configfile>" option.

Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|";
        msg: "mountd access";)
```

The text up to the first parenthesis is the rule header and the section enclosed in parenthesis is the rule options. The words before the colon in the rule options section are called option keywords. Note that the rule options section is not specifically required by any rule, they are just used for the sake of making tighter definitions of packets to collect or alert on (or drop, for that matter). All of the elements in that rule must be true for the indicated rule action to be taken. When taken together, the elements can be considered to form a logical AND statement. At the same time, the various rules in a Snort rules library file can be considered to form a large logical OR statement.

Again, there are a lot of possibilities which are explained in the *Snort Users Manual*. There is no point in learning the possibilities by heart but I advise you to look at the manual to get a global idea of what is possible.

Tripwire

What is it?

Tripwire is a cross platform software suite from a company by the same name. Tripwire monitors system changes.

All information in this chapter comes from *Tripwire's web-site* . Consult the site for more information.

Tripwire does not prevent an intruder or virus from attacking system files but detects intrusions when they occur. By comparing system files and directories against a previously stored "baseline" database, Tripwire finds any additions, deletions or changes to the specified properties. This enables the system administrator to determine the extent of the problem and begin the necessary damage control.

Installation

Linux Open Source Code for Tripwire can be found at *Source Forge: Project Tripwire*.

Since there is a Debian package for tripwire, I have chosen to install it the Debian way:

```
apt-get install tripwire
```

You will then get the following message:

```
Tripwire keeps its configuration in an encrypted database that is generated, by
default, from /etc/tripwire/twcfg.txt

any changes to /etc/tripwire/twcfg.txt, either as a result of a change in this
package or due to administrator activity, require the regeneration of the
encrypted database before they will take effect.

Selecting this action will result in your being prompted for the site key
pass-phrase during the post-installation process of this package.

Rebuild the Tripwire configuration file?
```

Choose to rebuild it (Yes). You will then get the following message:

```
Tripwire keeps its policies on what attributes of which files should be
monitored in an encrypted database that is generated, by default, from
/etc/tripwire/twpol.txt

Any changes to /etc/tripwire/twpol.txt, either as a result of a change
in this package or due to administrator activity, require the
regeneration of the encrypted database before they will take effect.
```

Selecting this action will result in your being prompted for the site key pass-phrase during the post-installation process of this package.

Rebuild Tripwire policy file?

Choose to rebuild it (Yes). You will then get the following message:

Tripwire uses two different keys for authentication and encryption of files. The site key is used to protect files that could be used across several systems. This includes the policy and configuration files.

You are prompted for this pass-phrase either because no site key exists at this time or because you have requested the rebuilding of the policy or configuration files.

Remember this pass-phrase, it is not stored anywhere!

Enter site-key pass-phrase

Just enter a phrase you can remember, such as "Be careful not to trip over the wire". You will then get the following message:

Please repeat the site pass-phrase to be sure you didn't mistype.

Repeat the site-key pass-phrase

Retype your pass-phrase. You will then get the following message:

Tripwire uses two different keys for authentication and encryption of files. The local key is used to protect files specific to the local

machine, such as the Tripwire database. The local key may also be used for signing integrity check reports.

You are being prompted for this pass-phrase because no local key file currently exists.

Remember this pass-phrase, it is not stored anywhere!

Enter local key pass-phrase

Just type another pass-phrase you can remember, such as "Another Tripwire phrase". You will then

get the following message:

Please repeat the local pass-phrase to be sure you didn't mistype.

Repeat the local key pass-phrase

Retype your pass-phrase. You will then get the following message:

Tripwire has been installed.

The Tripwire binaries are located in /usr/bin and the database is
located in /var/lib/tripwire. It is strongly advised that these
locations be stored on write-protected media (e.g. mounted RO floppy).
See /usr/share/doc/tripwire/README.Debian for details.

This concludes the installation of Tripwire.

Configuration

This section describes the files that influence the behavior of Tripwire.

The Tripwire Configuration File

The Tripwire configuration file is structured as a list of keyword-value pairs and may also contain comments and variable definitions. Any lines with "#" in the first column are treated as comments.

After installation, the file /etc/tripwire/twcfg.txt contains the following lines:

```
ROOT        =/usr/sbin
POLFILE     =/etc/tripwire/tw.pol
DBFILE      =/var/lib/tripwire/$(HOSTNAME).twd
REPORTFILE  =/var/lib/tripwire/report/$(HOSTNAME)-$(DATE).twr
SITEKEYFILE =/etc/tripwire/site.key
LOCALKEYFILE =/etc/tripwire/$(HOSTNAME)-local.key
EDITOR      =/usr/bin/vi
LATEPROMPTING =false
LOOSEDIRECTORYCHECKING =false
MAILNOVIOLATIONS =true
EMAILREPORTLEVEL =3
REPORTLEVEL  =3
SYSLOGREPORTING =true
MAILMETHOD   =SMTP
SMTPHOST     =localhost
SMTPPORT     =25
```

This is the plain text version of the file /etc/tripwire/tw.cfg. After modifying the plain text version, a new, encrypted version of that file must be generated by issuing the command **twadmin --create-cfgfile**.

Required Variables

The following variables must be set in order for Tripwire to operate:

POLFILE Default = /etc/tripwire/tw.pol
DBFILE Default = /var/lib/tripwire/$(HOSTNAME).twd
REPORTFILE Default = /var/lib/tripwire/report/$(HOSTNAME)-$(DATE).twr
SITEKEYFILE Default = /etc/tripwire/site.key
LOCALKEYFILE Default = /etc/tripwire/$(HOSTNAME)-local.key

Other variables

The following variables are not required to run Tripwire, but without them, the program's functionality will be limited. The values assigned during installation are listed.

Optional variables

EDITOR

Specifies an editor to be used in interactive modes. If EDITOR is not defined, and no editor is specified on the command line, using interactive modes will cause an error. Initial value: /bin/vi

TEMPDIRECTORY

This variable determines where Tripwire writes its temporary files. By default it is /tmp, which, due to the default permissions, can be very insecure. It is recommended that you use this configuration variable to provide tripwire with a secure place to write temporary files. The directory should have permissions set so that only the owning process can read/write to it, i.e., **chmod 700**. Initial value: /tmp

GLOBALEMAIL

This variable is set to a list of email addresses separated by either a comma (,) or semi-colon (;). If a report would have normally been sent out, it will also be send to this list of recipients. Initial value: none

LATEPROMPTING

Prompt for passphrase as late as possible to minimize the amount of time that the passphrase is stored in memory. If the value is true (case-sensitive), then late prompting is turned on. With any other value, or if the variable is removed from the configuration file, late prompting is turned off. Initial value: false

LOOSEDIRECTORYCHECKING

When a file is added or removed from a directory, Tripwire reports both the changes to the file itself and the modification to the directory (size, num links, etc.). This can create redundant entries in Tripwire reports. With loose directory checking, Tripwire will not check directories for any properties that would change when a file was added or deleted. This includes: size, number of links, access time, change time, modification time, number of blocks, growing file and all hashes. If the value for this variable is true (case-sensitive), then loose directory checking is turned on, and these properties will be ignored for all directories. With any other value, or if the variable is removed from the configuration file, loose directory checking is turned off. Turning loose directory checking on is equivalent to appending the following propertymask to the rules for all directory inodes: -snacmblCMSH. Initial value: false

SYSLOGREPORTING

If this variable is set to true, messages are sent to the syslog for four events: database initialization, integrity check completions, database updates and policy updates. The syslog messages are sent from the "user" facility at the "notice" level. For more information, see the syslogd(1) man page and the syslog.conf file. The following illustrates the information logged in the syslog for each of the four events:

Jun 18 14:09:42 lighthouse tripwire[9444]: Database initialized: /var/lib/tripwire/test.twd

Jun 18 14:10:57 lighthouse tripwire[9671]: Integrity Check Complete: TWReport lighthouse 20000618141057 V:2 S:90 A:1 R:0 C:1

Jun 18 14:11:19 lighthouse tripwire[9672]: Database Update Complete: /var/lib/tripwire/test.twd

Jun 18 14:18:26 lighthouse tripwire[9683]: Policy Update Complete: /var/lib/tripwire/test.twd

The letters in the Integrity Checking log correspond to the number of violations, maximum severity level and the number of files added, deleted or changed, respectively. With any value other than true, or if this variable is removed from the configuration file, syslog reporting will be turned off. Initial value: true

REPORTLEVEL

Specifies the default level of report produced by the **twprint --print-report** mode. Valid values for this option are 0 to 4. The report level specified by this option can be overridden with the (-t or --report-level) option on the command line. If this variable is not included in the configuration file, the default report level is 3. Note that only reports printed using the twprint --print-report mode are affected by this parameter; reports displayed by other modes and other commands are not affected. Initial value: 3

MAILMETHOD

Specifies the protocol to be used by Tripwire for email notification. The only acceptable values for this field are SMTP or SENDMAIL. Any other value will produce an error message. Initial value: SENDMAIL

SMTPHOST

Specifies the domain name or IP address of the SMTP server used for email notification. Ignored unless MAILMETHOD is set to SMTP. Initial value: mail.domain.com

SMTPPORT

Specifies the port number used with SMTP. Ignored unless MAILMETHOD is set to SMTP. Initial value: 25

MAILPROGRAM

Specifies the program used for email reporting of rule violations if MAILMETHOD is set to SENDMAIL. The program must take an RFC822 style mail header, and recipients will be listed in the "To: " field of the mail header. Some mail programs interpret a line consisting of only a single period character to mean end-of-input, and all text after that is ignored. Since there is a small possibility that a Tripwire report would contain such a line, the mail program specified must be able to ignore lines that consist of a single period (the -oi option to sendmail produces this behaviour). Initial value: **/usr/lib/sendmail -oi -t**

EMAILREPORTLEVEL

Specifies the default level of report produced by the **tripwire --check** mode email report. Valid values for this option are 0 to 4. The report level specified by this option can be overridden with the (-t or --email-report-level) option on the command-line. If this variable is not included in the configuration file, the default report level is 3. Initial value: 3

MAILNOVIOLATIONS

This option controls the way that Tripwire sends email notification if no rule violations are found during an integrity check. If MAILNOVIOLATIONS is set to false and no violations are found, Tripwire will not send a report. With any other value, or if the variable is removed from the configuration file, Tripwire will send an email message stating that no violations were found. Mailing reports of no violations allows an administrator to distinguish between unattended integrity checks that are failing to run and integrity checks that are running but are not finding any violations. However, mailing no violations reports will increase the amount of data that must be processed. Initial value: true

The Tripwire Policy File

The policy file describes system objects to be monitored by Tripwire and specifies which properties for each object should be collected and stored in the database file. Each object in the policy file is associated with a property mask, which describes what changes to the file or directory Tripwire should monitor, and which ones can safely be ignored. By customizing the various aspects of the policy file, the system administrator can very closely control how Tripwire checks the integrity of any system.

The basic components of policy files

Comments

In a policy file, any text following a "#", up to the next line break, is considered a comment.

Directives

Tripwire supports a small number of directives that allow conditional interpretation of the

policy file and certain diagnostic and debugging operations. The primary purpose of directives is to support sharing of a policy file among multiple machines. Directives use the following syntax:

@@ directive_name [arguments]

Where the directive name is one of the directives listed below:

```
@@section  # Designates a section of the policy file.
@@ifhost   # Allow conditional interpretation
@@else     # of the policy file.
@@endif
@@print    # Print a message to standard output.
@@error    # Print a message to standard output and then exit.
@@end      # Marks the logical end-of-file.
```

Variables

For user convenience, Tripwire's policy file supports variables for string substitution. Variables can be defined anywhere between rules. The syntax for variable definition is:

variable = value;

Variable substitution is legal anywhere that a string could appear. The syntax for variable substitution is:

$(variable)

Rules

Policy rules determine whether and to what extent Tripwire will check particular files and directories. There are two kinds of policy rules recognized by Tripwire:

1) Normal rules define which properties of a particular file or directory tree Tripwire scans.

2) Stop points tell Tripwire not to scan a particular file or directory.

The format for a normal rule is:

object_name -> property_mask;

The Format for stop points is:

! object_name ;

After installation, the plain text version of the policy file /etc/tripwire/twpol.txt contains the following lines which you should read at least once to get a global idea of how things are done. An excerpt:

```
...
# Global Variable Definitions

@@section GLOBAL
TWBIN = /usr/sbin;
TWETC = /etc/tripwire;
TWVAR = /var/lib/tripwire;

#
# File System Definitions
#
@@section FS

#
# First, some variables to make configuration easier
#
SEC_CRIT      = $(IgnoreNone)-SHa ;  # Critical files that cannot change
SEC_SUID      = $(IgnoreNone)-SHa ;  # Binaries with the SUID or SGID flags set
SEC_BIN       = $(ReadOnly) ;        # Binaries that should not change
...

#
# Tripwire Binaries
#
(
  rulename = "Tripwire Binaries",
  severity = $(SIG_HI)
)
{
  $(TWBIN)/siggen                   -> $(SEC_BIN) ;
  $(TWBIN)/tripwire                 -> $(SEC_BIN) ;
  $(TWBIN)/twadmin                  -> $(SEC_BIN) ;
  $(TWBIN)/twprint                  -> $(SEC_BIN) ;
}
...
```

Once the initial policy file has been generated, any changes should be made with the **tripwire --update-policy** command, rather than by simply overwriting the policy file with the **twadmin --create-polfile** command. This is an important distinction--when a new policy file is created, the Tripwire database must be re-initialized. If an intruder has modified files since the last integrity check, these changes will not be detected and will be included as part of the new "baseline" database.

The nmap command

What is it?

nmap is a network exploration tool and security scanner. It can be used to scan a network, determine which hosts are up and what services they are offering.

nmap supports a large number of scanning techniques such as: UDP, TCP connect(), TCP SYN (half open), ftp proxy (bounce attack), Reverse-ident, ICMP (ping sweep), FIN, ACK sweep, Xmas Tree, SYN sweep, IP Protocol and Null scan.

If you have built a firewall, and you check that no ports are open that you do not want open, **nmap** is the tool to use.

Using the nmap command

Assuming we have got a host fictitious.test and we want to see what tcp ports this host is listening to, this is done as follows:

```
# nmap -sT fictitious.test

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0

Nmap run completed -- 1 IP address (0 hosts up) scanned in 30 seconds
```

As you can see, this didn't work and since I'm very sure that the host is up, I can connect to it by means of **ssh**, I will issue the command again with the -P0 option:

```
# nmap -sT -P0 fictitious.test

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on fictitious.test (ip address):
(The 1545 ports scanned but not shown below are in state: filtered)
Port       State       Service
22/tcp     open        ssh
137/tcp    closed      netbios-ns
138/tcp    closed      netbios-dgm
139/tcp    closed      netbios-ssn

Nmap run completed -- 1 IP address (1 host up) scanned in 304 seconds
```

After this command, the ports are only tested for accessibility by means of the TCP protocol. Let's try the same command on the Microsoft web-site:

```
# nmap -sT www.microsoft.com

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
```

```
Interesting ports on microsoft.com (207.46.197.102):
(The 1544 ports scanned but not shown below are in state: filtered)
Port       State      Service
80/tcp     open       http
137/tcp    closed     netbios-ns
138/tcp    closed     netbios-dgm
139/tcp    closed     netbios-ssn
443/tcp    open       https

Nmap run completed -- 1 IP address (1 host up) scanned in 383 seconds
```

Note the difference: the machine fictitious.test is not running a webserver and Microsoft is (ports 80 and 443).

Have a look at nmap's manual page, there are a lot of command line options.

Keeping track of security alerts

What are they?

Security alerts are warnings about vulnerabilities in certain pieces of software. Those vulnerabilities can result in a decrease of your service level because certain individuals are very good at misusing those vulnerabilities. This can result in your system being hacked or blown out of the water.

Most of the time there is already a solution for the problem or someone is already working on one, as will be described in the rest of this section.

Bugtraq

What is it?

BugTraq is a full disclosure moderated mailing-list for the detailed discussion and announcement of computer security vulnerabilities: what they are, how to exploit them and how to fix them.

Where is it?

On the website *SecurityFocus* there is a link to mailing lists, one of which is Bugtraq. There also is a Bugtraq FAQ.

Subscribing to the Bugtraq mailing list

You can subscribe to the Bugtraq mailing-list by sending an e-mail message to bugtraq-subscribe@securityfocus.com. The content of the subject or message body do not matter. You will receive a confirmation request to which you will have to answer.

Unsubscribing from the Bugtraq mailing list

Send an e-mail message to bugtraq-unsubscribe@securityfocus.com from the subscribed address. The content of the subject or message body do not matter. You will receive a confirmation request to which you will have to answer.

If your email address has changed email listadmin@securityfocus.com and ask to be manually removed.

CERT

What is it?

The CERT Coordination Center (CERT/CC) is a center of Internet security expertise, at the Software Engineering Institute, a federally funded research and development center operated by Carnegie Mellon University. They study Internet security vulnerabilities, handle computer security incidents, publish security alerts, research long-term changes in networked systems and develop information and training to help you improve security at your site.

Where is it?

CERT maintains a website called *The CERT Coordination Center*.

How to subscribe to the CERT Advisory mailing list

You can subscribe to the CERT advisory mailing list by sending an email to majordomo@cert.org. In the body of the message, type "subscribe cert-advisory" without the single quotes.

How to unsubscribe to the CERT Advisory mailing list

To remove your name from the CERT mailing list, send email to majordomo@cert.org. In the body of the message, type "unsubscribe cert-advisory" without the single quotes.

CIAC

What is it?

CIAC is the U.S. Department of Energy's Computer Incident Advisory Capability. Established in 1989, shortly after the Internet Worm, CIAC provides various computer security services free of charge to employees and contractors of the DOE, such as: Incident Handling consulting, Computer Security Information, On-site Workshops, White-hat Audits.

Where is it?

There is a *CIAC Website*.

Subscribing to the mailing list

CIAC has several self-subscribing mailing lists for electronic publications:

CIAC-BULLETIN for Advisories, highest priority - time critical information, and Bulletins, important computer security information.

CIAC-NOTES for Notes, a collection of computer security articles.

SPI-ANNOUNCE for official news about Security Profile Inspector (SPI) software updates, new features, distribution and availability.

SPI-NOTES, for discussion of problems and solutions regarding the use of SPI products.

The mailing lists are managed by a public domain software package called ListProcessor, which ignores E-mail header subject lines. To subscribe (add yourself) to one of the mailing lists, send requests of the following form: subscribe list-name LastName, FirstName, PhoneNumber as the E-mail message body, substituting CIAC-BULLETIN, CIAC-NOTES, SPI-ANNOUNCE or SPI-NOTES for "list-name" and valid information for "LastName" "FirstName" and "PhoneNumber." Send to: ciac-listproc@llnl.gov.

You will receive an acknowledgment containing address and initial PIN, and information on how to change either of them, cancel your subscription or get help.

Unsubscribing from the mailing list

To be removed from a mailing list, send the following request via E-mail to ciac-listproc@llnl.gov: unsubscribe list-name.

Testing for open mail relays

What are they?

An open mail relay is a mail server that accepts SMTP connections from anywhere. This means that everyone can connect to port 25 on that mail server and send mail to whomever they want. If you are unlucky, other system administrators might add your server's IP address to their DENY/REJECT or equivalent list.

Testing for them

Testing if your mail daemon is an open mail relay is in fact quite easy. Telnet from a machine that should not be able to use your mail server as a relay to your mail server's port 25 and try to send an e-mail as shown below:

```
charon:~# telnet mail.home.nl 25
Trying 213.51.129.253...
Connected to mail.home.nl.
Escape character is '^]'.
220 mail2.home.nl ESMTP server (InterMail vM.4.01.03.00 201-229-121) ready Fri,
11 Jan 2002 17:19:14 +0100
```

```
HELO
250 mail2.home.nl
MAIL FROM: willem@test.bla.bla.bla
250 Sender <willem@test.bla.bla.bla> Ok
RCPT TO: willem@snow.nl
250 Recipient <willem@snow.nl> Ok
DATA
354 Ok Send data ending with <CRLF>.<CRLF>
FROM: willem@test.bla.bla.bla
TO: willem@snow.nl

Hahaha, open mail relay test
.
250 Message received: 20020111162325.GOJI8897.mail2.home.nl@snow.castel.nl
QUIT
221 mail2.home.nl ESMTP server closing connection
Connection closed by foreign host.
```

This worked because this is my ISP and I _do_ belong to the right domain. I tried it from a wrong domain, and I got no response whatsoever. You could use IPCHAINS, IPTABLES or some other sort of firewall software to tell your firewall to only forward the SMTP protocol packets to your mail server if they are coming from a certain range of IP addresses (for instance, the dynamic ones you have reserved for your PPP users). Also, most mail servers allow configuration settings to avoid acting as an open relay. Nowadays, this is the default behaviour for most mail server implementations.

Chapter 13. System Customization and Automation (2.213)

Prev Next

Chapter 13. System Customization and Automation (2.213)

Revision: $Revision: 1.4 $ ($Date: 2007/01/10 15:36:27 $)

This topic has a total weight of 3 points and contains the following objective:

Objective 2.213.1; Automating tasks using scripts (3)

Candidates should be able to write simple scripts to automate tasks using different common scripting languages. Tasks for automation include checking processes, process execution, parsing logs, synchronising files across machines, monitoring files for changes, generating and sending e-mail alerts and notifying administrators when specified users log in or out.

Key knowledge area(s):

Standard text manipulation software sucs as awk and sed
BASH
cron configuration files
at daemon usage
Remote copying software such as rsync and scp
Perl: Basic commands

The following is a partial list of the used files, terms and utilities:

perl
bash
awk
sed
crontab
at

This chapter discusses three main programs that work with Regular Expressions: Perl, sed and awk. Because Regular Expressions require a rather extended explanation and are

common in all three programs, the Regular Expressions are discussed first.

The chapter concludes with topics about rsync, crontab and monitoring your system, as listed in the lpic objectives above.

Regular Expressions

Revision: $Revision: 1.10 $ ($Date: 2007/01/11 08:31:04 $)

Resources: (none).

Further reading: Friedl01, manpages for the various commands.

Regular Expressions are, as you will read further on in this chapter, *the salt of all Unix systems*. Regular Expression patterns are used in **grep** and derivatives, Perl, **awk** and **sed**. In general, there are lots of similarities between the Regular Expression dialects used by each of these programs. When able to work with one dialect, it is quite easy to work with the other dialects too. There are, however, annoying and dangerous differences. In this section the commonalities will be described and the differences clarified. Unfortunately, portability is an issue: there are even different Regular Expression dialects for the same program on different operating systems. Linux Regular Expression support is less complicated, but even here differences exist. This point will gradually become clear in this section.

In keeping with common practice in Perl documentation, the term Perl will be used for the Perl language and **perl** will be used for the program.

Introducing Regular Expressions

A Regular Expression is a pattern. The pattern is *matched* against some text. The result: the pattern will either fit or not fit. There is a lot to say about this pattern language. The text is frequently called the *input*. On the command line or in scripts, it can originate from an input stream (*standard input*) or from a set of files. In these cases the *input* will almost always be of a line of text. When using a programming language that supports Regular Expressions, the pattern will probably be matched against the contents of a buffer.

Note

When using Regular Expressions on the command line, such as with grep, you must protect the Regular Expression from expansion by the shell. Putting a single (forward) quote on each side of the Regular Expression will do the job:

```
grep 'loo*ny' tune.texts
```

This (rather long) section will discuss Regular Expressions in detail. First, the Regular Expression language (including variants) is introduced. Next, Regular Expressions are applied using specific programs. So, take a deep breath and start reading.

Primitives and Multipliers

The parts of a Regular Expression most commonly used are the *primitive* and the *multiplier*. They frequently occur in pairs. An example of such a pair is:

```
E{4}
```

In this example, E is the primitive and {4} is the multiplier, meaning four times the thing before it. This Regular Expression will match four consecutive capital E characters somewhere in the *input*.

Given the Regular Expression above, as an example, and an input line such as the following:

```
I like lots of EEEE's
```

the Regular Expression will match the four E's in the input. Moreover, the complete line will match, since the Regular Expression fits somewhere in the line. A member of the **grep** family, such as **egrep** will show the input if the Regular Expression matches part or all of it. So, if **egrep** were to handle something like this

```
echo "Lots of EEEE's" | egrep 'E{4}'
```

then **egrep** would show the whole line.

A multiplier may be omitted, in which case a multiplier of *one time* is added automatically to each primitive. So the Regular Expression:

```
head
```

effectively means:

> one h, followed by one e followed by one a followed by one d.

Both primitives and multipliers are discussed below (the section called "Primitives" and the section called "Multipliers"). More facilities are available for Regular Expressions: anchoring, grouping and alternation. These are discussed later on.

Primitives

There are three types of primitives: a regular character, the dot placeholder and the character class. The character class is the most complicated of these.

A digit, letter or other character

Any regular character can be a primitive. This includes letters (like E) and digits. For instance, in the following example the capital E is the primitive:

```
E{4}
```

What is the primitive here:

```
4{2}
```

The correct answer is 4.

Other characters can also be used as primitives in a Regular Expression. For example:

```
ls -la | egrep ' {11}'
```

(the Regular Expression is a space followed by multiplier {11}). This will show lines containing at least 11 consecutive spaces.

Some characters have special meanings, such as the curly brace ({). These characters must be *escaped* (they are preceded by a backslash (\)) to remove the special meaning. This will be discussed further later on.

The dot: placeholder for any character

The dot has a special meaning. As a primitive, it can be a placeholder to allow *any character* in that position. For instance:

```
a.c
```

allows for one (remember the one-time multiplier being added?) a, followed by any single character, followed by one c. So

```
echo reading the abc | egrep 'a.c'
```

fits.

And what will this fit on?

```
a.{3}
```

The answer: one a followed by three arbitrary characters.

> **Note**
>
> To find a literal dot, use the \. primitive.

Character classes

A character class is a placeholder for a number of characters. It is more restrictive than the dot placeholder, since it allows you to specify the characters the placeholder should match.

Keep in mind that the character class is a *primitive*, it is a placeholder for *one* of the specified characters.

To set up a character class, start with a [(opening square bracket) and end it with] (closing square bracket). Any combination of *sets* and *ranges* can be specified between these square brackets.

A character *set* in a character class

To specify a set (a series) of characters in a character class, simply put them between the square brackets. So,

```
[aouiyA]
```

specifies a set consisting of an a (both lower and upper case), an o, a u or a y.

Remember that the character class is a primitive. That is, it can be multiplied:

```
[aouiyA]{2}
```

This specifies *two* characters, each of which may be one of the characters inside the square brackets.

The order of the characters in the set is irrelevant. That is, [abcABC] is the same as the [aAbBcC] character class. So,

```
echo reading the abc | egrep 'a[bB]c'
```

fits, as would

```
echo reading the aBc | egrep 'a[bB]c'
```

A character *range* in a character class

A character *range* consists of two characters separated by a minus sign. This specifies one digit:

```
[0-9]
```

This is one lower case letter:

```
[a-z]
```

Ranges can be combined:

```
[a-z0-9]
```

Some sets can be written as ranges. For example, the Regular Expression:

```
[ghiGHI]{2}
```

can also be written as:

```
[g-iG-I]{2}
```

It matches two of the indicated characters.

For now, the order is defined by the ASCII character set (or derivatives like iso8859-*x*). In the ASCII character set, letters and digits are not adjacent. That is why they need to be specified separately as ranges:

```
[a-z0-9]
```

More and more systems support so-called *locale* handling (such as the character set for a given country) of Regular Expressions. Arising character definitions like Unicode will probably change this.

Combinations of sets and ranges

Ranges and sets can be combined at will. This specifies one of the named vowels or a digit:

```
[aouiy0-9]
```

And this one matches three characters:

```
[A-Z][aouiy0-9][A-Z]
```

Which one? Answer: three characters, the first and the third an uppercase letter with a vowel or digit in the middle. In the following example the Regular Expression matches:

```
echo This is JaN | egrep '[A-Z][aouiy0-9][A-Z]'
```

> **Note**
>
> To include a minus sign in a character set, place it at the beginning or the end of the character class.

The inverted character class

It is often desirable to **exclude** certain characters from a character class. That's where the *inverted* character class comes in.

The *inverted* character class starts with a caret (^) as the first character after the opening square bracket ([) that starts the character class. The caret sign at the beginning inverts (negates) the meaning of the character class. Where

```
[0-9]
```

fits one digit,

```
[^0-9]
```

will fit *any character that is not a digit*.

This, of course, can be extended with every character class:

```
[^aeouy0-9]
```

will fit anything that is neither a vowel nor a digit.

> **Note**
>
> To include a caret sign in a character class, do not put it at the beginning.

POSIX character classes

The *POSIX* character classes form an extension to the character classes that were discussed above. They come in the following format:

[[:*keyword*:]]

The *keyword* is a word describing the class, for example, alnum or blank. Current implementations of GNU **grep** and **egrep**, as well as GNU awk (**gawk**), come with manpages that describe the POSIX character classes. Let's look at some examples.

Note

The outer square brackets are the delimiters of any character class. The inner square brackets and colons plus the *keyword* define the POSIX character set.

Table 13.1. Overview of character classes

Notation	Description
[[:alnum:]]	It fits exactly one alphanumeric character. It is the same as [a-zA-Z0-9]
[[:alpha:]]	This class matches any letter
[[:upper:]]	This class matches any uppercase letter (same as: [A-Z])
[[:lower:]]	This class matches any lowercase letter (same as: [a-z])
[[:digit:]]	This class matches any digit (same as: [0-9])

[[:xdigit:]]	This class matches any hexadecimal digit (same as: [0-9A-Fa-f])
[[:blank:]]	It matches a blank or a tab sign (a subset of the wider set called *white space*)
[[:space:]]	It matches each white-space character, including *space*, *tab*, *formfeed* and *carriage return*
[[:punct:]]	This class matches all punctiation characters

Multiple POSIX-character-set definitions can be combined in one character class:

```
[[:punct:][:space:]]
```

This one fits a character which is either a punctuation character (such as a semicolon or a plus sign) or a white-space character.

The POSIX character set specifiers can be combined with any other character-class element, as shown below:

```
[0-9]{4}[[:space:],\.:][0-9]{2}
```

In words: four digits, then one character that can be either a white-space character, a comma, a dot or a colon, followed by two digits.

Since the POSIX character class extensions are extensions to the regular character class, it can be inverted the same way:

```
[^[:digit:]]
```

This matches one non-digit. It is identical to:

```
[^0-9]
```

Multipliers

A multiplier (also known as *quantifier*) works in conjunction with a primitive. The multiplier is placed after the primitive. Together they form a specification of how many of which

characters are wanted.

Unfortunately, before multipliers can be explained in detail, the two main variants in Regular Expressions, most affecting the way multipliers are spelled, need to be introduced first. The *classic* Regular Expressions are the ones implemented in **grep** and **sed**. The modern, or *extended* Regular Expressions, on the other hand, are the ones used by **awk**, **egrep**, Perl and **flex**, to name a few. The multipliers shown so far were in the extended style. For this reason, **egrep**, was used in examples.

We will discuss 4 multipliers. They are the *, ? and + multipliers (and their classic counterparts *, \? and \+, yes, the * is the same in both cases).

You can also use the curly-brace multipliers we've already met. These are part of the POSIX standard. All relevant GNU programs have support for curly-brace multipliers on board; even versions for classic Regular Expression exist. In GNU awk (**gawk**) a special --posix option must be given to turn on support for curly-brace multipliers. Table 13.2, "Multipliers" at the end of this section will provide an overview of Regular Expression multipliers.

The * multiplier

The * is the oldest multiplier around. As far as I know, it is implemented in every Regular Expression language. It is the same in classic and extended variants.

The * means: *zero or more times*. So,

```
a*
```

means zero or more a's. This may not be that meaningful, since it is always true. In larger combinations, however, its use can be more meaningful:

```
ba*c
```

That is, one b, and one c, possibly with a's between them.

The ? multiplier

The ? multiplier means *zero or one time*, in other words, *present* or *not present*. The extended version is ?, the version to use in classic Regular Expressions is \?.

An example: in sgml/xml sections start with <section followed by a lot of things, and end with </section, also followed by other things. To fetch both lines, use

```
egrep '</?section' regexp.xml
```

The classic variant is

```
grep '</\?section' regexp.xml
```

The + multiplier

The + multiplier means *one or more times*. The extended version is +, the version to use in classic Regular Expressions is \+.

This (extended) Regular Expression

```
ba+c
```

means one b, followed by at least one a, followed by a c.

To find long, regardless of how may o's it contains, use the (extended) Regular Expression

```
lo+ng
```

The curly brace multiplier

In the extended variant, a normal set of curly braces is used, as in {4}. In the classic variant, a matching pair of \{ and \} (each curly brace is prefixed by a backslash) is used, as in \{4\}.

The curly-brace multipliers come in four variants (only the extended Regular Expressions will be shown). The general form is

{*m*,*n*}

In this, both *m* and *n* are numbers indicating the lowest (*m*) and the highest (*n*) number. It matches *at least m*, but *at most n* times. For instance, to match at least two, but at most four digits:

```
[0-9]{2,4}
```

To specify a minimum number of times, omit the *n* (but not the comma!):

{*m*,}

For instance, to match at least five digits:

```
[0-9]{5,}
```

And to show lines that are at least 78 characters long (show also line numbers), select

```
egrep -n '.{78,}' file1 file2 ...
```

To specify a maximum number of times, omit the *m* (but not the comma!):

```
{,n}
```

For instance, to match up to three digits:

```
[0-9]{,3}
```

In this case, the minimum will be zero.

> **Note**
>
> The example alone will not limit to three digits. It will just match up to three characters. If more numbers adjacent to these three are not wanted, anchors must be used - see the section called "Anchors".

To match something no more and no less than *n* times, specify only *n* (do not use a comma):

```
{n}
```

To match exactly four capital alphabetic characters, for instance:

```
[A-Z]{4}
```

Support and portability. Curly-brace multipliers are supported by Perl. In GNU awk, support must be enabled explicitly with the --posix option; other awk implementations might not (yet) support these multipliers (called *interval expressions* in awk terminology). GNU egrep will support them. GNU grep supports the classic version. In other versions of (e)grep, support may be lacking. GNU sed supports the classic implementation. There are implementations of sed that do not support curly-brace multipliers at all. In yet other implementations of sed, extended Regular Expressions can be selected. POSIX standards are said to require curly-brace multipliers, so support is expected to be implemented elsewhere over time. Table 13.3, "Portable multipliers" provides an overview.

Multiplier overview

Table 13.2, "Multipliers" shows all multipliers and variants together.

Table 13.2. Multipliers

extended RE	classic RE	meaning
*	*	zero or more times
?	\?	zero or one time
+	\+	one or more time
{*m*,*n*}	\{*m*,*n*\}	at least *m*, but maximally *n* times
{,*n*}	\{,*n*\}	up to *n* times
{*m*,}	\{*m*,\}	at least *m* times
{*m*}	\{*m*\}	exactly *m* times

extended RE:	Perl, GNU awk with --posix, GNU egrep
classic RE:	GNU grep, GNU sed

Note

GNU awk without --posix and other awk implementations currently support only the *, + and ? multipliers.

Portability. If your Regular Expressions should also be used on non-GNU platforms portability becomes an issue. Old grep's will probably not support the \+ and \? multipliers. Curly-brace multipliers (called *interval expression* in awk terminology) will probably not be supported in other awks (though they are reported to be part of the POSIX standard). Even GNU awk (**gawk**) does not enable interval expressions by default: they can only be enabled by invoking either the --posix or the --re-interval option. Table 13.3, "Portable multipliers" lists multipliers that you can use safely across platforms. But even here, exceptions do exist.

Table 13.3. Portable multipliers

Perl	awk, egrep	sed, grep	meaning
*	*	*	zero or more times
?	?		zero or one time
+	+		one or more times
{*m*,*n*}			at least *m*, but maximally *n* times

{,*n*}			up to *n* times
{*m*,}			at least *m* times
{*n*}			exactly *n* times

Anchors, Grouping and Alternation

Anchors

Sometimes a Regular Expression should only match if a pattern matches at a certain position, for instance, at the beginning of the input or at the beginning of a word. This can be achieved by adding an anchor to your Regular Expression.

Anchors are also called *zero width assertions* because they fit between or adjacent to something.

There are two types of anchors: anchors specifying a beginning or ending of a line and anchors specifying a word boundary. Both types will be discussed.

Begin and end anchors

When an anchor is used in a Regular Expression, the Regular Expression will only match if the string is in the position specified by the anchor.

Generally, newlines are stripped before the match is attempted. This is true for at least awk, sed and all grep variants. But, in Perl, you must either remove the newline using the chomp operator or include it in the Regular Expression. Furthermore, if the so-called *multi-line mode* is turned on, the meaning of begin and end anchors changes altogether in Perl. More in the section called "Perl Regular Expressions".

Portability. Begin and end anchors are supported in all Regular Expression variants and use the same notation everywhere.

Let's look at the begin and end anchors in more detail.

The ^ begin anchor

The ^ (caret) is the anchor that specifies the beginning of the input. If you attach ^ at the beginning of your Regular Expression, matches will then only be found at the beginning of the input.

In the example below there is a match:

```
echo Once upon ... | grep '^Once'
```

There is, however, no match in this one:

```
echo He said: Once ... | grep '^Once'
```

The $ end anchor

The $ (dollar sign) is the anchor that specifies placement at the end of the input. The $ is typically attached to the end of the Regular Expression. The Regular Expression will then only match strings at the end of the input.

In the next example all lines that end in a slash (that is, containing directory names) will match:

```
ls -lF | grep '/$'
```

In this example, there will not be a match, since **pwd** does not put a slash at the end of its output:

```
pwd | grep '/$'
```

Matching complete text

By using the begin and end anchors together, you can create a Regular Expression that matches the entire input, not just some part of it:

```
^[0-9]+$'
```

specifies that the complete input should consist of digits and nothing else. So, if the input contains one non-digit, the Regular Expression will not match.

In the following example, **egrep** will *not* show empty lines or lines that only contain spaces:

```
egrep -v '^ *$' listofnames
```

So, the lines that will be shown contain at least one character that is not a space.

Word-boundary anchors

Word-boundary anchors fit between a word and something else. In most Regular Expressions word is defined as a series of alphanumeric characters. So, a word boundary fits between an alphanumeric character and a non-alphanumeric character. Furthermore, a

word-boundary anchor will also fit between an alphanumeric character and the beginning or end (where ^ and $ would match).

There are two variants of word-boundary anchors. The first is \b, which matches on either side of a word. The second variant consists of two anchors: \< and \>. The \< anchor fits either between a non-alphanumeric character and an alphanumeric character or before the beginning of a line if the line starts with an alphanumeric character (like ^). The \> anchor fits either between an alphanumeric character and a non-alphanumeric character or after an alphanumeric character at the end of a line (like $). The spelling of word-boundary anchors is the same everywhere.

Portability. Word-boundary anchors are not supported widely, but can be found in all GNU implementations of the programs. The only implementation you can really be sure of is \b in Perl.

Let's look at word-boundary anchors in more detail.

The \b word-boundary anchor

The \b word-boundary anchor fits on either side of a word. This one matches:

```
echo The one and only | grep '\bone'
```

This one does not:

```
echo gone with the wind | grep '\bone'
```

The \b anchor can also be used at the end of a word. This one does not match:

```
echo printed onesided | grep 'one\b'
```

A more complicated example is:

```
\b[0-9]{6-10}\b
```

This will only match if the input contains a series of six, seven, eight, nine or ten digits.

The \< and \> word-boundary anchors

The \< and \> word-boundary anchors fit, respectively, on the left and right word boundaries. They can be used together, but this is not mandatory.

In:

```
echo The onesided page is gone | grep '\<one\>'
```

the Regular Expression will not match.

Anchor overview

Table 13.4, "Anchors" provides an overview of anchors and their support.

Table 13.4. Anchors

anchor	awk	egrep	Perl	grep, sed	meaning
^	yes	yes	yes	yes	beginning
$	yes	yes	yes	yes	ending
\b	no	yes	yes	yes	word boundary
\<	no	yes	no	yes	left word boundary
\>	no	yes	no	yes	right word boundary

Notes
^ and $ in Perl can have a slightly different meaning in conjunction with newlines.
GNU awk (gawk) does support \< and \>
GNU grep and egrep do support word boundaries. Other grep's might not do this.

Grouping

Grouping is making part of a Regular Expression a separate entity. This is done by putting that part inside parentheses, like:

```
[0-9]:([0-9]:)
```

In the above, the second [0-9]: is inside parentheses, hence it is grouped.

> **Note**
>
> The group (the part between the parentheses) itself must be a correct Regular Expression.

Why would you use grouping? There are three important reasons:

1. a multiplier can be applied to a group. Discussed next.
2. the input the group matches can be re-used (so-called backreferences). Discussed next.
3. to use alternation in part of the Regular Expression. Will be discussed later in the section called "Grouping and Alternation".

Unfortunately, the difference between classic and extended Regular Expressions pops up again here. Table 13.5, "Grouping operators" shows when to use which spelling.

Table 13.5. Grouping operators

spelling	program
(...)	awk, Perl, egrep
\(...\)	grep, sed

Support and portability. Grouping support is present in Perl together with various backreference mechanisms. Grouping is also supported by all **awk** variants, but backreferences are not. GNU **grep**, **egrep** and **sed** support both grouping and backreferences. Every program that allows grouping also allows multiplication applied to a group.

Applying a multiplier to a group

A multiplier can be applied to a group. Simply put the multiplier right after the closing parenthesis of a group. To find a representation of time in the typical format two digits - colon - two digits - colon - two digits (such as 14:10:17), for example, use the following (extended) Regular Expression:

```
([0-9]{2}:){2}[0-9]{2}
```

To find all words with "ing" twice (such as singing), use this extended Regular Expression:

```
(ing){2}
```

If a group is not followed by a multiplier, then multiplication by one is chosen automatically.

Grouping and backreferences

The part of the input on which a group matches comes available for re-use. A special mechanism for backward referencing, called a *backreference* is available for this. The backreference corresponding to the first group in a Regular Expression is indicated with "\1" (a backslash and the digit one), the one corresponding to the second group \2, and so

on. If, for example, you are looking for the occurrence of four identical digits, you might use:

```
([0-9])\1\1\1
```

Suppose this is tried on:

```
The 1-2-3 of 4444
```

First, the RE finds 1 as the first digit. There is a group around it, so the value is saved. In the Regular Expression there is a \1 after the group. This tells the Regular Expression engine to use the same value (digit 1) again. This fails, since there is a dash after the 1. This sequence is repeated for the 2 and the 3, but both fail. However, when it finds the 4 it than finds the other three 4s, so the Regular Expression matches at last.

In an earlier paragraph I needed to find words that contain twice the same three characters, like singing. Here is how I found words like that:

```
egrep '(...)\1' /usr/share/dict/american-english
```

(I know, I should have used [A-Za-z]{3} instead of the dots. Since there are only letters in the file, dots work as well).

Extracting the matching part (or even subparts) using grouping is very important in Perl. More in the section called "Perl Regular Expressions".

Alternation

Using alternation, one can specify either/or constructions in a Regular Expression.

Again, the difference between classic and extended Regular Expressions pops up. Table 13.6, "Alternation operator" shows when to use which spelling.

Table 13.6. Alternation operator

spelling	program
|	awk, Perl, egrep
\\|	grep, sed

As an example, suppose a file contains phone numbers of mobile phones. Wanted: only the numbers that apply to The Netherlands. In the Netherlands, mobile phone numbers start with 06. When called from outside the Netherlands, this should be 00316. This is not true in

all countries, so some numbers start with +316. Here is the Regular Expression to handle all three possibilities:

```
egrep '06|00316|\+31' phonenumbers
```

Grouping and Alternation

Alternation can be used inside a group. Then, the alternation applies only to the part in the group.

Suppose the mobile phone numbers should be at the beginning of the input line. Then we'll we need to use an anchor. But if we use an anchor followed by the existing Regular Expression (^06|...), then the anchor applies only to the first number. The following is the correct solution:

```
egrep '^(06|00316|\+31)' phonenumbers
```

Another example: Find numbers in the range 1 .. 12. Leading zeroes are not allowed. Solution (extended Regular Expression):

```
\b([1-9]|1[0-2])\b
```

In words: one digit in the range 1-9 or two digits in the range 10-12.

Special characters

Two topics are discussed here:

1. setting or removing the special meaning of a character
2. white space

Characters and combinations with a special meaning

Rule 1. If a character has a special meaning by itself, it will lose this special meaning if a backslash is put before it.

The dot by itself has a special meaning, but \. will match a literal dot, as in

```
[0-9]+\.[0-9]{2}
```

Also, to find a literal star it needs to be prefixed with a backslash. The following will match zero or more digits or stars.

```
[0-9\*]*
```

To find a phone number starting with a literal plus followed by one or more digits, use this **egrep** command line:

```
egrep '\+[0-9]+' holidayaddress.mail
```

Rule 2. If a backslash and a character together have a special meaning, then omitting the backslash will remove the special meaning.

In some Regular Expression variants, the \b has a special meaning as word boundary. When the b is used alone, it just means the letter b.

```
\bback
```

In classic Regular Expressions, the \+ has a special meaning (a multiplier). A + without a backslash matches a literal +:

```
grep '+[0-9]\+ holidayaddress.mail
```

which is, of course, completely the opposite of the earlier extended Regular Expression example.

White space

A white-space character is either a space character or a tab character. Sometimes the definition is extended with a formfeed character and a carriage-return character.

In classic Regular Expressions, handling white space, especially tab characters, can be a burden. In extended Regular expressions, this is done comfortably. Perl is the most luxurious. All will be discussed next.

Classic Regular Expressions

In some classic Regular Expression variants (notably sed), there is no easy primitive for a tab character, let alone for white space. It is probably best to refrain from using classic Regular Expressions for tab handling. Two workarounds are available.

Tab characters

One way to handle white space is to create a character class consisting of a space and a literal tab character. That is the [, followed by a space, followed by a real tab character, followed by].

Note

This will not work on the command line. Most shells intercept the tab character for other purposes.

In an editor you can usually type the tab character as ^I (control-I). In some editors, tab has a special function, so something must be done first. In emacs, for example, you must type ^Q^I to get a literal tab character.

Note

Make sure your editor does not change tabs automatically when saving the file (check with od, for example).

Replacing tabs beforehand

The second workaround is to replace tab characters before they reach your classic-Regular-Expression program. Using tr (transliterate) is one option.

```
... | tr '\011' ' ' | sed ...
```

Here, any tab character (denoted by \011) is changed to a space character before the information is sent to sed.

Note

tr can only read from *standard input* (e.g., a pipe).

POSIX white space

If your software supports POSIX character-classes (Linux aims towards POSIX compliance), then you probably have the [[:blank:]] and [[:space:]] character classes available. In the following example the [[:blank:]] character class is used in a Regular Expression that matches a #, possibly preceded white-space characters, at the beginning of the line. Since -v is used, **grep** will show all lines on which the Regular Expression does *not* match:

```
grep -v '^[[:blank:]]*#' whatever.conf
```

The resulting output consists of all lines that do not start with a # and do not start with one or more white-space characters and a #. This is a way to show a file without the comments in it.

Extended Regular Expressions

The extended Regular Expressions have the \t notation for a tab character. With that, it is easy to build a white-space character class:

```
[ \t]
```

This means: either a space or a tab character. Applied:

```
[0-9]{1,2}[ \t]+[A-Z]
```

That is, one or two digits, followed by one or more white-space characters and an uppercase letter.

POSIX character classes are used exactly the same way as in classic Regular Expressions (above).

If you can use Perl for white-space handling, you are lucky: Perl has the \s primitive that matches a white-space character. Applied:

```
[0-9]{1,2}\s+[A-Z]
```

Or, to see lines starting with a # possibly preceded by white space:

```
^\s*#
```

For more on Perl Regular Expressions, see the section called "Perl Regular Expressions".

Regular Expressions in sed

This section will only describe what Regular Expressions look like in **sed**. How and where Regular Expressions are applied is described in the section called "Using **sed**".

The **sed** program uses the classic Regular Expression version. Probably because of a fear of breaking old software, few new features have been added, not even in GNU sed.

Remember that sed is line oriented. Operations work on a line that has been read from a file or *standard input*. The *global* modifier for the *substitute* command works on a line basis. When global is turned on, the replacement is done as often as possible in each line. When *global* is kept off, then the replacement will work once *in each input line*.

As described earlier, parsing **white space** is a burden in sed. Luckily, GNU sed 3.02 supports the POSIX character-set primitives, so [[:space:]]\+ can be used to match one or more white-space characters.

Regular Expressions in awk

This section will only describe what Regular Expressions look like in **awk**. How and where Regular Expressions are applied is described in the section called "Using **awk**".

There are at least three variants of awk for Linux. Of those, GNU awk (**gawk**) is the most sophisticated. The gawk distribution comes with great documentation.

All awk variants accept the extended Regular Expressions described earlier, but, currently, without the curly-brace multipliers (called *interval expressions* in awk terminology). Keep in mind that **gawk** has some extensions that the other awk variants do not accept, such as POSIX character-class extensions and the \< and \> word anchors. The *interval expressions* are supported in **gawk**, but are not enabled by default for portability with other awk implementations. To enable these, use either the --posix or the --re-interval option.

You might have wondered why most awk variants do not have support for word anchors. Since each awk split its input into *fields* (which are normal words by default) these fields can be checked with normal begin and end anchors. The following checks, for example, if the second field (word) begins with an uppercase letter:

```
echo The Ape | awk '$2 ~ /^[A-Z]/ {print $2 ": starts upcase"}'
```

Creating a **white-space** character class in awk is easy: [\t] (as was shown earlier). In **gawk**, the POSIX white-space character classes can be used instead. This is, of course, not portable to other awks.

Perl Regular Expressions

Further Reading: lots of Perl documentation, e.g. Wall01. See **perldoc perlre** on your computer for more information about Perl Regular Expressions.

Perl has the most powerful Regular Expression variant. If at all possible, use Perl for your Regular Expression tasks.

> **Note**
>
> This section will only describe the Perl Regular Expressions and how to use them in Perl. The Perl language itself is described in the section called "Using Perl".

The Perl Regular Expression language is basically the extended Regular Expression language introduced earlier. There are, however, some subtle differences and extensions, which will be discussed below. The section on Perl Regular Expressions will end with a presentation on how to use grouping to extract parts of a Regular Expression match.

Perl character class extensions

Perl has extra primitives, many of which turn out to be very handy in everyday use. Table 13.7, "Extra primitives in Perl" shows a selection.

Table 13.7. Extra primitives in Perl

in Perl	description	same in awk
\d	a digit	[0-9]
\D	a non-digit	[^0-9]
\s	a white-space character	[\t\f\r]
\S	a non-white-space character	[^ \t\f\r]
\w	a "word" character	[a-zA-Z0-9_]
\W	a non-"word" character	[^a-zA-Z0-9_]

These primitives can be used both as a standalone primitive and inside a character class.

> **Note**
>
> The Perl definition of \w is not the same as an alphanumeric character. Instead, it is defined as an alphanumeric character or an underline character ([a-zA-Z0-9_]).

Difference: the dot

The dot means: any character, *but not the newline character*. In the *single line mode*, which is not discussed further in this book, the dot will also match the newline.

Special characters

In Perl, it is easy to distinction between characters that need to be prefixed with a backslash to remove a (possible) special meaning and those that do not, namely \w characters versus \W characters.

\w **characters.** Any primitive that matches \w is a normal character without a special meaning. The letter a, for example, falls into this category.

\W **characters.** Any character that matches \W may have a special meaning. Prefix it with a backslash to make sure it loses that special meaning. If it does not have a special meaning, putting a backslash before it will do no harm. The + character, for example, falls in this category.

Anchors: line boundaries

Line-boundary handling is very sophisticated in Perl. Consider for example, that the meaning of ^ and $ changes when *multi-line mode* is used. For more information you might start with **perldoc perlre**, then read other documentation.

Newlines are not removed automatically. When reading lines (from a file or information stream) Perl does not chop off newlines automatically. You can either remove the newline before the match is applied, or you can include the newline in your Regular Expression. Here is an example that removes the newline using the chomp operator:

```
while (<>)  # read line, put it in $_
{
   chomp;  # remove newline from $_

   print $_,"\n" if (/9{2}$/); # show $_ if it ends in 99
}
```

If you keep the newline character and want to match the end of line you must include the newline in your Regular Expression:

```
/9{2}\n$/
```

This matches if there are two nines and a newline-character at the end of the input.

Anchors: word boundaries

As far as we know, the anchor \b originated in Perl, and is now available in some other Regular Expression variants, as shown earlier. Perl has the \b, but also the opposite \B. Table 13.8, "Perl (non-)word boundary anchors" shows both.

Table 13.8. Perl (non-)word boundary anchors

primitive	description
\b	word boundary (between \w and \W)
\B	not a word boundary

Extracting matching parts

Perl has two ways to re-use the matching parts (or even subparts). The first is done by

reading some special pseudo variables, the second involves grouping.

Method 1: using $‘, $& and $’

After a successful match, the three related pseudo variables $‘, $& and $’ will contain the following:

$&

The $& pseudo variable (correct spelling: a dollar sign and an ampersand character) contains that part of the input that the Regular Expression precisely matched.

$‘

The $‘ pseudo variable (correct spelling: a dollar sign and a backward quote a.k.a. *backtick*) contains the part of the input *before* the matching part.

$’

The $’ pseudo variable (correct spelling: a dollar sign and a forward single quote) contains the part of the input *after* the matching part.

Note

Be sure to confirm that the Regular Expression really matched. Otherwise, the values of $‘, $& and $’ may be random.

An example:

```
$_ = 'This is a nice book';

if (/\w+ic\w+/)
{
   print
      "Input:           \"$_\"\n",
      "Part before match: \"$`\"\n",
      "Matching part:    \"$&\"\n",
      "Part after match:  \"$'\"\n";
}
```

The output of this program part is:

```
Input:           "This is a nice book"
Part before match: "This is a "
Matching part:    "nice"
```

```
Part after match:  " book"
```

Note that the contents of $` has an extra trailing space and that of $' an extra leading space.

Method 2: grouping

In the Perl Regular Expression language, one can use grouping to extract matching parts or even subparts from the input.

Suppose the input is:

```
Today is  7 January 2002, a monday.
```

The following Perl Regular Expression matches the complete date in the input:

```
\b\d{1,2}\s+[A-Za-z][a-z]+\s+\d{4}\b
```

In Perl, you can also group the day, the month and the year parts:

```
\b(\d{1,2})\s+([A-Za-z][a-z]+)\s+(\d{4})\b
```

When this Regular Expression is matched against the input, three pseudo variables, namely $1, $2 and $3 will be filled-in with the corresponding parts of the date. The information corresponding to the first group (seen from the left) is inserted in $1, the second in $2, and so on:

```
$_ = "Today is 7 January 2002, a monday.";

if (/\b(\d{1,2})\s+([A-Za-z][a-z]+)\s+(\d{4})\b/)
{
   print "Day:   $1\n",
         "Month: $2\n",
         "Year:  $3\n";
}
```

Nested groups. Groups can be nested:

```
\b((\d{1,2})\s+([A-Za-z][a-z]+)\s+(\d{4}))\b
```

Information corresponding to the utmost group (here: the whole date) gets into $1. The parts inside that group fall into $2, $3 and so on. To see this working, use something like this:

```
$_ = "Today is 7 January 2002, a monday.";

if (/\b((\d{1,2})\s+([A-Za-z][a-z]+)\s+(\d{4}))\b/)
{
   print "Date:  $1\n",
         "Day:   $2\n",
         "Month: $3\n",
         "Year:  $4\n";
}
```

Alternate method 2: using the binding operator

There is an alternative to the use of the pseudo variables ($1 etc.). The parts of the input that correspond to the groups in the Regular Expression can be written in an array. In the program part below, these subparts of the input are stored in the array @results:

```
my $input = "Today is 7 January 2002, a monday.";

my @result = $input =~ /\b((\d{1,2})\s+([A-Za-z][a-z]+)\s+(\d{4}))\b/;

if (@result)    # if RE fits
# etc
```

It is important to realize that

```
$input =~ /RE/
```

actually is a statement that returns an array. To capture this, put an array assignment before it:

```
@result = $input =~ /RE/
```

This really is correct Perl! Remember that the =~ operator is called the *binding* operator. It is not an assignment.

This program part puts everything in perspective:

```
my $input = "Today is 7 January 2002, a monday.";

my @result = $input =~ /\b((\d{1,2})\s+([A-Za-z][a-z]+)\s+(\d{4}))\b/;

if (@result)    # if RE fits
{
```

```
    for (my $i = 0; $i < scalar @result; $i++)
    {
        print "Field $i: $result[$i]\n";
    }
}
```

Elements of the array will be filled in the same order the pseudo variables were assigned. The first element, $results[0], will get the complete date, $results[1] will get the day, $results[2] the month and $results[3] the year.

Prev | Next

Security tasks (2.212.6) | Home | Using Perl

Using Perl

Revision: $Revision: 1.12 $ ($Date: 2007/01/10 16:22:01 $)

Resources and further reading: PerlRef02; PerlRef03; PerlRef04; Till01; **man** pages for the various commands; **perldoc** documentation, e.g. **perldoc perl**, **perldoc perlre**.

Corresponding to common practice in Perl documentation, we will use Perl for the Perl language and **perl** for the program.

Writing simple Perl scripts

Perl is an acronym that stands for *Practical Extraction and Report Language*. It is a language optimized for scanning text files, extracting information from those text files and printing reports based on that information. It can also be used for many system-management tasks. The language is intended to be practical (easy to use, efficient, complete), rather than beautiful (tiny, elegant, minimal). Perl combines features of **C**, **sed**, **awk** and **sh**. If you have a problem that you would ordinarily solve using **sed**, **awk** or **sh**, but it exceeds their capabilities or must run a little faster, and you can't or don't want to write in **C**, then Perl may be for you. There are also translators to turn your **sed** and **awk** scripts into Perl scripts. When you need powerful Regular Expression support, Perl will probably be the best option.

Perl's expression syntax corresponds closely to C expression syntax. Perl does not limit the size of your data, the only restriction is the size of your memory. Recursion is of unlimited depth. Tables used by hashes (previously called associative arrays) grow as necessary to prevent degraded performance. In Perl, you can use sophisticated pattern-matching techniques to scan large amounts of data quickly. Although optimized for scanning text, **perl** can also deal with binary data, and can make **dbm** files look like hashes. Setuid and setgid **perl** scripts are safer than C programs through a data-flow-tracing mechanism (taintmode) that closes a number of security holes.

Perl is an interpreted language, which means that there is no explicit compilation step. The **perl** processor reads its input file, converts it to an internal form and executes it immediately. Internally, however, **perl** compiles the program and only executes it if the compilation was successful.

A number of interesting features are:

- No need to explicitly declare variables.
- The type of a variable (scalar, array or hash) is indicated by a prefix character.
- No need to define the size of strings or of arrays before using them.
- All variables have predictable default values.
- A very rich set of pattern-matching operations that make it very easy to locate and process patterns of text.
- A complete set of arithmetic capabilities.
- A very complete set of built-in functions.

Perl basics

Warning

Perl is a very powerful and rich language. It goes far beyond the scope of this book to try to teach you all about it. The objectives require that you know just enough about Perl to enable you to write *simple* Perl scripts. Therefore, we will cover the basics and leave it up to you to explore further. It is assumed the reader has at least a nodding acquaintance with at least one programming language, preferably **C** or **awk**, and therefore knows concepts like variables, arrays, branching, looping, control statements and the like. After all, you are an LPIC-1 alumnus ;-)

First steps

perl will use standard input by default and look there for statements it should execute. Thus, to execute Perl statements, you could start up the **perl** program just by typing in its name[19]:

```
$ perl
```

The **perl** interpreter starts up, and silently waits for input. It is possible to type in a number of Perl statements next. Every command in Perl should end in a semicolon; to forget one is a common error, both for experienced and novice users. Thus, a Perl program could look like this:

```
print "hello world\n";
exit 0;
```

To make **perl** interpret these statements, terminate the input by typing **Ctrl**+**D** on the next line. Now **perl** will check the statements you typed, check for errors and execute them. If all went well, you will see the following on your screen:

```
hello world
```

Most of the time, you will want to execute a program that consists of statements in a file. It is possible to start up **perl**, and tell it to execute statements from within a file:

```
$ perl perlfile
```

All script files (shell, Perl) are handled the same way: statements are contained in files which have the execution bit set (**chmod +x**). If the first line of an script contains a hash and an exclamation mark -- #! -- as its first two characters, this denotes that the location of an interpreter follows. The shell will start that interpreter and feed the script to it. For example:

```
#!/usr/bin/perl

perl statements
```

Everything after the first line is Perl speak. Lines beginning with a hash (#) are comments. Remember that statements in Perl should end in a semicolon. Thus, a Perl script could look something like this:

```
#!/usr/bin/perl

statement1;
# a comment line
statement2;
statement3;  # end of line comment
```

Perl statements can also be put in a *block*. A block consists of a opening curly brace, some statements and a closing curly brace. For instance:

```
if ($x > 5)
{
   statement5;
   statement6;
}
```

The block is used in several places: each if, for example, **must** be followed by a block and statements in a subroutine definition will be contained in a block.

Once you have written a Perl program this way, and have set the execution bit on it, you can simply run it by typing the name of the file. Again, **perl** will check your code first and generate error messages if it finds mistakes.

Printing

In many of our examples, we will use the built-in print function. print takes a list containing one or more arguments and prints them. If a file handle (file handles) is placed between the print keyword and the arguments, the arguments are sent to the corresponding file. By default, standard output will be used. The **print** function does not add newlines or other formatting.

Formatting can be added by including escape characters, or you can use the printf function. C programmers will feel at ease with this function right away. The first string argument to **printf** is a format string, which is a mixture of normal characters and format directives. Normal characters are simply output *as is*. Format directives begin with a "%" and are instructions on how to output the next argument from the list of arguments following the format string. Format directives may contain width and precision specification, instructions for left or right justification, and type of data. Please see the Perl documentation (start with **perldoc perl**. An example:

```
$v = 10;
$h = 33.6;
printf( "The area of a %f x %f rectangle is %10.3f\n", $v, $h, $v * $h );
```

will print:

```
The area of a 10 x 33.6 rectangle is    336.000
```

Variables

You can store all types of data in variables. An example:

```
#!/usr/bin/perl

$x = 1;
$y = 2;
$z = 3;

print "$x + $y = $z\n";
```

Any variables in quotes will be evaluated *before* the print prints out the results. So, running this program results in the following output:

```
1 + 2 = 3
```

Scope of variables

By default, all variables in Perl are global variables; that is, they are accessible from every part of the program. You can create private variables called lexical variables at any time with the `my` operator. Their scope is restricted to the block or file they are declared within. A block is denoted by a set of curly braces (the block that defines a subroutine or an "if" statement or a specially created block):

```
{
   my($x, $y); # private variables for this block

}
```

You can force yourself to use only lexical variables with the following pragma:

```
use strict;
```

When using this pragma, unwanted use of a global variable will be caught. For instance:

```
#!/usr/bin/perl -w

use strict;

$verse = "Some text";
```

This results in a compile-time error. To fix this, use `my $verse` to make `$verse` a local variable (this time at file level) or use the `use vars ...` pragma to define `$verse` as a known global variable.

In Perl, the first character of a variable name or variable reference denotes what kind of value the variable may hold. Here are the most common kinds of variables:

Table 13.9. Variable types in Perl

$var	names a variable that holds a scalar data type (integer, floating point or string).
@var	names a variable that holds an array of scalars or list.
%var	names a variable that holds a hash of scalars.

The above table names three *different* variables. In other words, variable $x is a different variable than @x.

Scalar variables

A Perl scalar variable can be an integer, floating point or a character string. Integer literals are written without a decimal point or an "E" exponent. Examples of integer literals are:

```
12354
-179
0x32    # hexadecimal
027     # octal
```

The third number in the list is a hexadecimal (base 16) constant; its value is 50 decimal. The fourth number in the list is an octal (base 8) constant because it begins with a "0"; its value is 23 decimal.

Floating point literal numbers are written with a single decimal point and/or an "E" exponent. Example floating point literals are:

```
12.3
-6.23E27
3E7
.1
```

String literals are enclosed in either single or double quotes:

```
"The world\n"
```

Another example:

```
'no expansions'
```

Single-quoted strings. In a single-quoted string no expansion will be done. In the following the $ is just a dollar sign and \n are the two characters backslash and n:

```
print 'Some $verse here \n';
```

The output will be:

```
Some $verse here \n
```

Commands between backticks (back quotes) and escape sequences will not not expanded.

Double-quoted strings. Material in double-quoted strings is subject to expansion. Among these are variables and escape sequences. An escape sequence is a combination of a backslash and character(s). The combination will have a special meaning. The newline character, for example, will be written as the \n combination in a double-quoted string:

```
my $verse = 'hallo there';
print "Some $verse here\n";
```

This will result in

```
Some hallo there here
```

The last character of the output is a newline. If you want $, @ or % to appear literally in a string delimited by double quotes, escape them with a single backslash. Another example is the double-quote character itself: to get a double quote in a string surrounded by double quotes, use the \ for escaping the double-quote. Table 13.10, "Escape characters in Perl" shows a list of well-known escape sequences.

Table 13.10. Escape characters in Perl

\t	tab
\n	newline
\r	return
\f	formfeed
\b	backspace
\a	alarm (bell)
\e	escape character
\033	octal character
\"	double quote
\\	literal backslash
\c	c, where "c" is a normal character

For example, suppose we had assigned $t = 'test'. Then:

```
$x = '$t'
```

would assign the literal string $t to $x, where

```
$x = "St"
```

would assign the literal string test to $x. Some example string literals are:

```
"Hello World!"
'A bit longer but still a nonsensical string'
''
""
```

When a variable contains a scalar literal, its name always begins with a "$". An example:

```
$length = 12.3;
$width = 17;
$area = $length * $width;       # $area should be 209.1
```

In the above, the value of $length was a floating-point number, the value of $width was an integer. The result of the calculation would be floating point.

An example using strings:

```
$colour = "blue";
$shade = "dark";
$description = $shade . " " . $colour
```

Here, the scalar variable $description will have the value "dark blue" ('.' is the string-concatenation operator).

Arrays

An array is a singly dimensioned vector of scalar quantities. Individual elements of the array are accessed by a small integer index (or subscript). Normally, an array containing "n" elements has indices starting at 0 and going to n - 1, inclusive. In contrast to many other programming languages, it is possible to have array literals (sometimes called lists). Examples of array literals are:

```
()              # the empty list, or array
(1,2,3)           # an array of three integers
(1,'fred',27.1,3*5)
```

The last example is of any array containing 4 elements. The second element (index==1) is a string and the fourth element (index==3) is an integer of value 15.

When a variable references an array of scalars, its name always begins with a "@". For

example:

```
@one = (1, 2, 3);
@two = (4, 5, 6);
@both = (@one, @two);
```

At the end of this, @both will be an array of six elements. The expression "(@one, @two)", in this context, effectively joins the two smaller arrays end-to-end.

A surprising feature

the fact is, that to reference one of the elements of an array, whose name starts with "@", you are referring to a scalar datum -- and thus the name must begin with "$".

You also must enclose the subscript or index in square brackets following the name. For example:

```
@vec = (3,4,5);
$sum = $vec[0] + $vec[2];
$vec[1] = $sum;
```

At the end of this, $sum will have the value of 8, and the array @vec will have the value (3,8,5).

Determining the size of an array

For every array @foo, there is a scalar variable $#foo that gives the index-id of the last element of the array. So, after execution of:

```
@vec = (5,10,15);
```

The value of $#vec is 2.

A more common way to do this is by using the automatic conversion from array to scalar. When an array is assigned to a scalar the scalar will get the number of elements in the array. The two following lines do exactly this:

```
my $size = @somearray;
my $size = scalar @somearray;
```

The scalar command makes the conversion even more visible.

Multi-dimensional arrays

Perl does not support multi-dimensional arrays. From a technical point of view, there is no need for multi-dimensional arrays - memory can be regarded as a one-dimensional array of bytes - but often the multi-dimensional notational closer matches reality. Think, for example, of programming a board game where it would be much more convenient to think in terms of "rows" and "columns", instead of terms like "offset counting from square one".

There are a number of ways to simulate multi-dimensional arrays using Perl. One of them uses *references*. A reference is a scalar that refers to an entire array, list or other variable type. To create a reference to a variable, you put a backslash in front of it. So, \@a denotes a reference to the array a. A reference is a scalar and therefore can be stored as an element of an array or list. Thus, it is possible to store a reference to an entire array in one element of another array. In the ASCII art below, an example is given. Each line represents an array (from left to right @a, from top to bottom @b and from front to back @c). A dash denotes an element in the array. The "X", for example, denotes element 4 in array @c (remember, we start counting at zero).

```
0 - - 3 - - - - - - a
    |    c
    |   /
    |  (X)  $c[4]
    |  /    $b[7]->[4]
    | /     $a[3]->[7]->[4]
    |/
    7
    |
    |
    b
```

To find the element (X) you can use three methods:

- The direct notation $c[4]: Denotes that (X) is element #4 of @c, or
- Using the notation $b[7]->[4]: Start at array @b, which denotes to use element #7 in array @b, which is a reference to array @c, and take the 4th element to find (X), or
- Using the notation $a[3]->[7]->[4]: Start at array @a, take element #3, which is a reference to array @b, of which element #7 is a reference to array @c, of which you take element #4.

Hash Variables

When a variable references a hash of scalars, its name always begins with a "%". For example:

```
%box = ("len", 100, "height", 40, "width", 20, "colour", "blue");
```

This can also be written as follows:

```
%box = ("len" => 100, "height" => 40, "width", 20, "colour" => "blue");
```

Both assign a hash with four elements to variable %box. The indices (or keys) of the elements are the strings "len", "height", "width" and "colour" and the values of those elements are 100, 40, 20, and "blue", respectively.

> **Elements are scalar data!**
>
> When we refer to elements, we are referring to scalar data and thus the variable reference must start with "$".

The index delimiters of '{}' denote that we are accessing a hash (just as the index delimiters '[]' denote access to an array). In our example, to get the height, use:

```
print "height is ", $box{'height'};
```

The output will be

```
height is 40
```

```
$box{'volume'} = $box{'width'} * $box{'length'} * $box{'height'};
```

This assigns a fifth element to hash %box.

User Subroutines

Defining a subroutine

Perl has the ability to use user-defined subroutines or functions. The name of such a subroutine consists of letters, digits and underscores, but can't start with a digit. It is defined by the sub keyword, followed by a name and a block:

```
my $n = 0; # file-wide variable $n

...
```

```
sub marine
{
  $n += 1;
  print "Hello, sailor number $n!\n";
}
```

Subroutine definitions can be anywhere in your program text. Subroutine definitions are global; without some powerful trickiness, there are no private subroutines. If you have two subroutine definitions with the same name, the later one overwrites the earlier one.

Calling a subroutine

Invoke a subroutine from within any expression by using the subroutine name, sometimes preceded with an ampersand:

```
&marine;  # says Hello, sailor number 1!
&marine;  # says Hello, sailor number 2!
&marine;  # says Hello, sailor number 3!
&marine;  # says Hello, sailor number 4!
```

The initial ampersand is often optional. It is mandatory, however, if you are defining a function of your own that has the same name as one of the *built-in* functions of Perl (this is a bad idea anyway):

```
sub chomp
{
   print "bit more than I could chew";
}

&chomp; # prevents calling the internal chomp function
```

You may omit the ampersand if there is no built-in with the same name and if the definition of the function precedes the calling of the function:

```
sub multiply
{
   $_[0] * $_[1];
}

...

my $mult = multiply 355, 113;
```

Passing arguments to subroutines

A special array @_ contains the arguments passed to the subroutine on entrance. It is a good idea to save the arguments in a local variable:

```
# show elements of array by index
sub showarray
{
  my @list = @_;

  for (my $i = 0; $i < @list; $i++)
  {
     print "index $i value $list[$i]\n";
  }
}
```

This is how the subroutine is called:

```
my @rows = (24,30,12,34);

...

showarray @rows;
```

The output is:

```
index 0 value 24
index 1 value 30
index 2 value 12
index 3 value 34
```

When passing argument(s) to a subroutine, you actually pass an array. In fact, you are passing *one* array. When you want to pass separate entities (e.g., two arrays), you must pass references to these entities instead and dereference these in the subroutine itself. This is left as an exercise to the reader.

Returning from a subroutine

The result of the last executed statement of a subroutine will be the return value of the subroutine. This subroutine will return the sum of the first two arguments:

```
sub add
{
```

```
    $_[0] + $_[1];
}
```

You can also make the subroutine return by using the return keyword:

```
sub divide
{
    if ($_[1] == 0)
    {
        return 0;
    }

    # otherwise: return the division
    $_[0] / $_[1];
}
```

Operators and (internal) functions

Operators allow a combination of constants and variables. There are many common operators, however we will only discuss the most important ones. The precedence of the operations are largely intuitive. Precedence of operations can be overridden by the use of parentheses. When in doubt, use parentheses to prevent ambiguity as a last resort. Check the Perl syntax first.

Numerical operators. Numerical operators expect numbers as arguments and return numbers as results. The basic numeric operators are: + - * / **, (the last one means exponentiation). Additionally, these operators may be combined with the "=" equal sign, to form C-style operators such as += -= *= **=:

```
$n  = 8;
$n **= 3;  # same as: $n = $n ** 3;
print $n;  # 512 (8 * 8 * 8)..
```

Among the built-in numeric functions are: cos, sin, exp, log and sqrt. Additionally, the increment-by-one (++) and decrement-by-one (--) operators are used in Perl in the same way as they are used in C. As in the C language, their placement is of importance:

```
$m = 1;

print $m++ . " ";   # print value first, than increment..
print $m   . " ";   # print incremented value ..
print ++$m . " ";   # increment first, then print new value..
print $m   . "\n"; # print value;
```

This would result in the printing of the range: 1 2 3 3

Automatic type conversion

numeric operators and functions expect numeric arguments. If they are given a string argument, they convert to numeric automatically.

String concatenation operator. To add strings, use the string concatenation operator: . (a dot). It denotes string concatenation:

```
$i = 'hi';
$o = 'ho';
$song = $i . $o;
```

Variable $song will now contain hiho.

Boolean operators. There is no special data type that signifies Boolean values. There are just values that mean false: the zero or null values. More specifically, they are:

```
""     null string
"0"    string
0      integer
0.0    float
()     empty array or list
```

A variable can be set to undef, meaning that it is completely undefined. Everything that is not zero, null, empty or undef will mean true. There are a number of operators that explicitly produce Boolean results: these always produce 0 to indicate false and 1 to indicate true. The most common are:

```
||  or
&&  and
!   not
```

The first two are *short-circuit* operators; the right operand is not evaluated if the answer can be determinded solely from the left operand. The "not" operator ! reverses the value of its only operand: true will become false, vice versa.

Numeric Comparisons. For comparing numbers, you can use:

```
<  less
<= less or equal
== equals
!= not equal
>= bigger or equal
>  bigger
```

Strings are not numerics

If you compare *strings* using these *numeric* comparison operators, the strings are first converted to numeric quantities and the numeric quantities compared.

String Comparisons. The commonly used ones:

```
lt  lesser than
le  lesser or equal
eq  equals (identical)
ne  not equal
ge  greater or equal
gt  greater than
```

The operators eq and ne are used for direct string comparisons. The others are used mainly in sorting. To perform more sophisticated string comparisons in Perl, use pattern matching. Perl permits the use of powerful pattern-matching operators: =~ and !~ for *does match* and *does not match* respectively.

Pattern matching with Regular Expressions

Regular Expressions are probably the most important feature of Perl. Regular Expressions in general are described in the section called "Regular Expressions". Be sure to read the part about Perl Regular Expression extensions (see the section called "Perl Regular Expressions"). This section shows how to **apply** Regular Expressions in Perl.

To match a Regular Expression against a variable, a statement must be created consisting of

- the variable: $some
- the *binding operator* =~
- a Regular Expression enclosed in slashes: /^[A-Z]/

A comparison statement can look as follows:

```
$some =~ /^[A-Z]/
```

This is not complete: the result of the comparison must be handled as well. In the following example the Boolean result of the above statement is evaluated (with an if statement). The result is either **true** or **false**. If **true** (if the contents of $some starts with an uppercase letter), the block is entered:

```
if ($some =~ /^[A-Z]/)
{
   # match: do something
}
```

If both variable and *binding operator* are omitted, the comparison is done against the contents of the so-called *default variable* $_:

```
if (/^[A-Z]/)
{
   # RE matches on $_: do something
}
```

Of course, if the Regular Expression matches, pseudo variables like $&, $1, etc. can be inspected, as described in the section called "Perl Regular Expressions".

There is also the reverse of the *binding operator*: the !~. Its result is **true** if the Regular Expression does not match the contents of the variable. In the example below, the block is entered when the contents of $some does not start with an uppercase letter:

```
if ($some !~ /^[A-Z]/)
{
   # if no match: do something
}
```

When matching against $_ the not operator ! can be used instead:

```
if (!/^[A-Z]/)
{
   # RE does not match on $_
}
```

Another way to check if a variable matches is to inspect the results of the match. This will only work if *grouping* (see the section called "Perl Regular Expressions") is used. The following is an example:

```
my @results = $input =~ /^([A-Z])([a-z]+)/;
```

```
if (@results)
{
   # RE matched!
   print "1st part: $results[0], 2nd part $results[1]\n";
}
```

File handles

Perl programs may need to refer to specific input or output streams or files. The variables that name them are, by convention, always rendered in uppercase. Output files can be explicitly created by the open function. For example, to open a file named prog.dat you would use:

```
open(OUT, ">prog.dat") || die "$0: open prog.dat for writing failed: $!";
```

This associates the file handle OUT with the file. The leading ">" signifies that the file will be emptied (if it did exist) or created. Use ">>" to instead *append* to the file.

> **Note**
>
> You must check if the open succeeds and catch possible errors. One of the possibilities is the use of die as shown above; $! will contain the error.

Now, you can use the file handle to write something to the file:

```
print OUT ("A test");
```

> **Note**
>
> When using print or printf and a handle **do not put a comma after the handle**.

This will open and empty the file and will write "A test" to it. *There is no comma between the file handle and the first string argument*.

Standard input is opened automatically and can be referenced via the predefined handle called "STDOUT". STDOUT will be used by printf and print if no file handle was specified. Another automatically opened handle is STDERR (*standard error*).

A file can be closed using the close keyword:

```
close(OUT);
```

File handles can also be used when reading from a file.

```
my $file = 'listOfNames';
open(IN, "<$file") || die "$0: unable to open $file for reading";
```

The initial < can be omitted.

To read from the file, use the file handle surrounded by < and >. There is, however, something to watch out for: scalar versus array context. If the handle is read in scalar context, exactly one line is read:

```
my $line = <IN>;
```

When this is used again, the next line is read and so on. In *array context* however, all lines will be read at once. The following, for example, will do exactly that:

```
my @allLines = <IN>;
```

The method of reading input line-by-line is often used in combination with a while keyword:

```
while (<IN>)
{
   # most recent line read in $_
   print $_;
}
```

This example reads each line and prints it, until no more lines can be read.

Don't forget to close the handles again:

```
close(IN);
```

A predefined file handle is STDIN for *standard input*. It is opened for reading. It can be used like any other handle.

A commonly used construction is <>. It operates as follows:

- If arguments are passed to the program (@ARGV is inspected), each of these arguments is considered to be a filename. Every <> reads one line until all files have been read.
- If, on the other hand, no arguments are found (@ARGV is empty), <> will be the same as <STDIN>.

In applications it might be used like this:

```
while (<>)
{
   ...statement(s)...
}
```

This will read either from named files (given as arguments) or from *standard input*. While inside the while loop, a special predefined variable $ARGV contains the name of the file that is being read (*standard input* is indicated by a dash (-)).

Process handles

A very powerful Perl facility is the *process handle*. It works exactly like a file handle but refers to a process instead. Process handles can be opened for reading or for writing.

The following code will open a WHO handle that is associated with a running **who** program.

```
open (WHO, "who|") || die "cannot start who: $!";

my @whoLines = <WHO>;

close (WHO);

print "There are currently ", scalar @whoLines, " users logged in\n";
```

The | symbol indicates that this is a process to be opened. By placing the | at the end, we tell Perl that we want to read from the process.

Of course, there are also writable process handles. A typical application might be writing a message (collected earlier) to a mail program:

```
open (MAIL, "|mutt -s \"log report\" $mailto") ||
   die "cannot start mutt: $!";

print MAIL @messageLines;

close (MAIL);
```

This time the | is at the beginning, indicating that the handle should be open for writing. The message is sent after the close statement has been processed.

Loops/repetition

Perl supports a number of loop constructs. Let's study the following example, in which we also recapitulate a number of the other things we've learned so far. The numbers in the left margin are for reference purposes only and do not belong to the code.

```
 /-------------------------------------------------------------------
 |
1 |  open ( PW, "</etc/passwd") || die "Eh..? Can't read /etc/passwd?";
2 |  while ( @a = split (':', <PW>) ) {
3 |    foreach $x ( @a ) { print $x . " "; }
4 |    print "\n";
5 |  }
6 |  close( PW );
 |
```

In line 1, the main logic functionality is determined by a *boolean expression*: the || ('or' operator) with two operands. The operands are functions in this case, but that is all perfectly legal in the Perl language. Remember that booleans act as short-circuit operators: the right operand is not evaluated if the answer can be determined solely from the left operand. Therefore, if the open succeeds, the die function will never be called. When open fails, the first part of the boolean expression is false and, therefore, the second part of the expression should be parsed (the die function will be called) and the script will terminate. For now, let's assume that the function opened the password file successfully. The file handle PW is now associated with it.

Line 2 contains a "while" loop. A "while" statement uses a test expression as its argument and executes its block every time the expression evaluates to true. A test expression is evaluated *before* every iteration, so the block may get executed 0 times. The core of the test expression is the split function. The split will take a string (its second argument) and split it into chunks, delimited by one of the characters as specified in the first argument (in our case, a colon). In our example, split fetches the input string directly from the file using the file handle "PW". The test expression evaluates to true as long as the array "@a" is filled by the split, which is the case as long as split is able to read from the file.

Line 3 gives an example of the foreach loop. The foreach loop provides a handy way to cycle through all elements of an array in order. It takes 2 arguments: the variable where an element of the array should be put in and the name of the array within parentheses. The loop cycles through all values in the array (effectively: all fields in the line from the password file) and stashes each field in $x, one field per iteration. The value is printed next.

Line 4 just prints a newline after each line that was read and parsed from the password file.

Line 5 terminates the "while" loop and

Line 6 closes the file associated with the file handle "PW".

Another example further clarifies the use of the foreach loop and introduces the *range* operator (three dots in a row) and the concept of lists:

```
@a = ("a"..."z","A"..."Z");
foreach $n (@a) {
   print $n;
}
```

In the first line, the array @a is filled with the elements of a list. A list is simply a listing of values. In this case, the list is formed by specifying two *ranges*, namely the range of all letters from "a" to "z" and the second range of all letters from "A" to "Z". The array @a will contain 52 elements. The foreach loops through all values in the array, one per iteration, and prints the resulting value to *standard output* (since we did not specify a file handle).

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The for loop is the third type of loop. It is identical to the C "for" construct. The general structure is:

```
for( init ; test ; incr ) block
```

A for statement can easily be rewritten using while:

```
my $i;                          my $i = 0;
for($i = 0; $i < 20; $i++)      while ($i < 20)
{                               {
   print "\$i: $i\n";                print "\$i: $i\n";
                                $i++;
}                               }
```

The test is performed before every iteration, so the block may be executed 0 times. Here, for example, is a way to print a table of squares, using the **printf** function:

```
for($n = 1; $n <= 10; ++$n)
{
   printf("%d squared is %d\n", $n, $n * $n);
}
```

Logic/branching

Perl supports the if statement:

```
if( $le < 10.0 )
{
   print("Length $le is too small\n");
}
```

Each test expression must be followed by a statement block. Blocks do not have to end with a semicolon. The following is functionally equivalent to our example above:

```
print( "Length $le is too small!\n" ) if ($le < 10.0);
```

There are "else" and "elsif" keywords too:

```
if( $le < 10.0 )
{
   print( "Length $le is too small!\n" );
}
elsif( $le > 100.0 )
{
   print( "Length $le is too big!\n" );
}
else
{
   print( "Length is just right!\n" );
}
```

Note that if, elsif and else *must* be followed by a block, even for one statement.

You can even use short-circuit boolean operators, as in:

```
$l < 10.0 && print("Length $le is too small!\n");
```

This kind of expression is particularly useful when you have to take an exceptional action after something (such as opening an input file) fails:

```
open(IN,"<foo.dat") || die("Unable to open foo.dat: $!");
```

written equivalently, but less readably, as:

```
die("Unable to open foo.dat: $!") if !open(IN,"<foo.dat");
```

Perl taint mode

perl automatically enables a set of special security checks, called *taint mode*, when it detects its program running with differing real and effective user or group IDs. In other words: when a program has either its setuid or setgid bit set (or both)[20].

You can also enable taint mode explicitly by using the *-T* command line flag:

```
#!/usr/bin/perl -w -T
```

Th -T flag is strongly suggested for server programs and any program run on behalf of someone else, such as CGI scripts. Once taint mode is on, it's on for the remainder of your script.

While in this mode, **perl** takes special precautions, called *taint checks*, to prevent both obvious and subtle traps. Some of these checks are reasonably simple, such as verifying that path directories aren't writable by others; careful programmers have always used checks like these. Other checks, however, are best supported by the language itself. As a result of these checks set?id **perl** programs are more secure, than similar C programs.

Note

Using taint mode does not take the responsibility away from the programmer. It just *helps* the programmer. For instance, if a program allows some user input from a web page, **perl** tests if the program checks this data, regardless of the quality of the test.

You may not use data derived from outside your program to affect something else outside your program -- at least, not by accident. All command-line arguments, environment variables, locale information (see the **perllocale** manual page), results of certain system calls (readdir(), readlink(), the variable of shmread(), the messages returned by msgrcv(), the password, gcos and shell fields returned by the getpwxxx() calls), and all file input are marked as "tainted".

Tainted data may not be used directly or indirectly in any command that invokes a subshell, nor in any command that modifies files, directories or processes.

Exception

If you pass a list of arguments to either system or exec, the elements of that list are NOT checked for taintedness. For example, assuming $arg is tainted, this is allowed (alas):

```
exec 'sh', '-c', $arg;      # Considered secure... :-(
```

> The programmer knows whether or not to trust $arg and, so, whether to take appropriate action.

Any variable set to a value derived from tainted data will itself be tainted, even if it is logically impossible for the tainted data to alter the variable. Because taintedness can be associated with particular scalar values, some elements of an array can be tainted and others not.

Perl modules

Perl scripts often use Perl modules. Great many Perl modules are freely available. A module provides a way to package Perl code for reuse. Many modules support object-oriented concepts. Modules are, in fact, *packages* that follow some strict conventions. Therefore, packages are explained first.

A package is a set of related Perl subroutines in its own namespace. Perl modules are often implemented in an Object Oriented way, thereby hiding the nitty-gritty details of the implementation of the module and presenting the user of the module with a small, clear interface.

A package starts with the package statement. The following example is the first line of CGI.pm:

```
package CGI;
```

This sets the namespace in CGI.pm to CGI.

When you write a Perl script, you will work in the default namespace called main. You can switch to another namespace by using another package statement.

A module is a package designed for reuse. However, modules are contained in files ending in the .pm extension (perl module) and are located in one of the Perl library directories. To make use of a module, the use keyword is used with the name of the module as argument:

```
use CGI;
```

If the module name contains two adjacent colons, the first part indicates a subdirectory. Thus, File::Basename refers to a module file called Basename.pm in the subdirectory named File in (one of) the library directories.

The predefined array @INC will list the directories where **perl** will start looking for modules. For instance, if directory /usr/lib/perl5 is in @INC, then **perl** will look for /usr/lib/perl5/File/Basename.pm when looking for the File::Basename module. The @INC can be easily extended.

For instance, to use a locally developed module in the directory /home/you/lib, use:

```
push(@INC,'/home/you/lib');
```

The **perl** -I flag can also be used:

```
#!/usr/bin/perl -w -I/home/you/lib
```

To make **perl** use a Perl module use the use keyword:

```
use File::Basename; # Look for File/Basename.pm.
```

You can nest deeper than a single directory. Just replace each double-colon with a directory separator.

CPAN

There are hundreds of free modules available on the *Comprehensive Perl Archive Network*, or CPAN, a set of Internet servers located throughout the world. It consists of about 100 sites that archive the same content. Many countries have at least one CPAN mirror, and their number is growing. Available modules include support for access to Oracle and other databases (DBI); networking protocols such as HTTP, POP3 and FTP and support for CGI.

CPAN offers various options to search for modules by author(s), category, name or date. Once you have found something interesting, you can download the module. You will have a compressed tarfile that contains the archive. You'll need to decompress the tarfile and untar it. Next, you can issue the commands:

```
# perl Makefile.PL
# make
# make test
```

If the test is successful, you can issue a make install to install the module. Make sure you have the appropriate permissions to install the module in your Perl 5 library directory. Often, you'll need to be root. That's all you need to do on Unix systems with dynamic linking[21].

Andreas Königs' CPAN module (CPAN.pm) is designed to automate the **make** and **install** of Perl modules and extensions. It includes some searching capabilities and knows a number of methods to fetch the raw data from the Net. Modules are fetched from one or more of the mirrored CPAN sites and unpacked in a dedicated directory. Most Perl installations already have this module pre-installed, but if not, you may want to download and install it first. Additionally, to allow you to perform extended searches for modules, there is another module available: CPAN::WAIT. It's a full-text search engine that indexes all documents

available in CPAN authors' directories. CPAN::WAIT uses a special protocol that resembles NNTP. CPAN::WAIT tries to access so-called "*wait servers*" and can do so using either a direct connection or over an http proxy.

The CPAN module normally is operated from within an interactive shell. That shell will allow additional search commands if the CPAN::WAIT module has been installed.

The initial configuration will be started automatically when the shell is started for the first time. To start the CPAN shell, type:

```
# perl -MCPAN -e shell;
```

The first time you start up this command, it will create some administrative files in which your preferences are kept. You can either choose to configure manually or let the module guess at the proper values (which often works out fine). Manual configuration will ask questions about the amount of disk space to use, the locations of some external programs the module wants to use, whether or not module dependencies should be resolved automatically, the location of your proxy servers (if any) and possible extra parameters you'd want to specify for **make**, etc. Also, the module will try several methods (if necessary) of gaining access to a CPAN site and may ask you for your location and your favorite *wait server*.

Perl on the command line

Perl can be easily invoked from the command line.

```
perl -e 'perl commands'
```

An easy way to find the hexadecimal value of a decimal number, for example, is:

```
perl -e 'printf "%x\n", 26;'
```

The output will be:

```
1a
```

The -n flag is very helpful on the command line. Suppose you want to use something like this:

```
while (<>)
{
   print "    :S_";
}
```

Everything but the print statement can be eliminated by using the -n option:

```
perl -ne 'print "    :$_";' inputs
```

This example will read lines from the file inputs. For each line that was read, output will be printed consisting of four spaces, a colon (:) and the original input line (expanded from $_).

---

[19] assuming **perl** was properly installed and the command occurs in your PATH

[20] The setuid bit in Unix permissions is mode 04000, the setgid bit mode 02000

[21] On systems that have a statically-linked perl (and the module requires compilation), you'll need to build a new Perl binary that includes the module

---

Writing Bourne shell scripts

Revision: $Revision: 1.8 $ ($Date: 2007/01/10 16:22:09 $)

Resources and further reading: Sayle98; **man bash**.

Being an LPIC-1 alumnus (or at least someone with equivalent knowledge), you are already familiar with the shell. By "*shell*", we mean a shell compliant to the IEEE POSIX Shell and Tools specification (IEEE Working Group 1003.2), for example, the Bourne shell or **bash**. In this section, we do not focus on the command-line interface, but on the writing of shell *scripts*. A shell script is just a set of shell commands, grouped into a file and executed by the shell. A very basic script could read like this:

```
#!/bin/sh
echo "Hello, world!"
exit 0
```

You can type this in using your favorite editor. After editing and saving it under some name - lets assume hello - you can execute it by starting up a shell with the program name as argument, for example:

```
$ sh hello
Hello World!
$ _
```

Alternately, you can set the execution bit on the program by issuing the command

```
$ chmod +x hello
```

Having done that, you can execute the script by simply typing its name:

```
$ ./hello
Hello World!
$ _
```

Note that the script should either be in a directory that is in your PATH, or you should specify its exact location, as we did in our example. Within the shell, the hash (#) is used to indicate a comment - anything on the same line after the hash will be ignored by the shell:

```
#!/bin/sh

# This program displays a welcoming text and exits.
# It was written for demonstration purposes.
echo "Hello, world!"
exit 0
```

Also, note that a script should pass an exit code to the outside world. By convention, the value 0 signals that the script ran correctly, a non-zero value signifies that some error occurred.

Variables

A variable within a shell script is just some data referenced by name. The data is always stored as a string of alpha numerical data. The shell does not work with concepts like "integers", "reals" or "floating point variables", everything is a string. Shell variable names may consist of upper- or lowercase letters, digits and the underscore. White space is not allowed in variable names. Some variables are predefined by the shell - a number of them are listed later on. All others can be user defined.

To set a variable within a script an equal sign is used:

```
A_VARIABLE=a_value
```

White space is *not* allowed anywhere in this construct. If you want to assign a string to a variable that contains white space, you either have to prepend it with a backslash, or put the string within double or single quotes, e.g:

```
A_VARIABLE="a value"
```

You can de-reference the variable name ("fetch its contents") by using a dollar-sign in front of the name of the variable. For example, to print the contents of the variable set in our previous example, you could use:

```
echo $A_VARIABLE
```

Sometimes, we will need to delimit the variable name itself to prevent confusion. Suppose we have a variable named A and another one named AA - the following statement can mean either "print the contents of the variable named A followed by the literal string 'A'" or it could mean "print the contents of the variable named AA":

```
echo $AA
```

To prevent uncertainty, use curly braces to delimit the variable name:

```
echo ${A}A  # $A and string "A"
echo ${AA}  # $AA
```

If a variable is not set, the NULL value is returned by default if the variable is de-referenced. However, the shell provides a number of notations that allow substitutions if the NULL value was found:

- ${varname:-*value*} will return the value of the variable varname if it is not NULL, otherwise the value *value* will be used;
- ${varname:=*value*} returns the value of the variable varname if it is not NULL; otherwise, the value *value* is used **and** the variable will be set to *value*;
- ${varname:?*value*} returns the value of varname if it is not NULL; otherwise, *value* is written to *standard error* and the script is exited. If *value* is omitted, then the message "parameter: parameter NULL or not set" is written to *standard error* instead;
- ${varname:+*value*} will substitute *value* if the variable contents are not NULL; otherwise, nothing is substituted.

Here are some examples of the use of substitutions: suppose we have a script that needs the variable MAILTO set. If we want the script to exit if that variable was not set, we could use:

```
MAILBOX=${MAILTO:?"MAILTO needs to be set"}
```

Or we might decide that the MAILBOX defaults to "root", like this:

```
MAILBOX=${MAILTO:=root}
```

Variables values may be de-referenced and reset at will. But a variable can be declared to be read-only, using the **readonly** command:

```
$ READONLY="Concrete"
$ readonly READONLY
$ echo ${READONLY}
Concrete
$ READONLY="Brick"
bash: READONLY: readonly variable
```

The previous example was run using **bash**, the default shell on Linux systems. It is fully compatible with the original Bourne shell.

The shell uses a number of internal variables. The values for these are generally set at login by parsing the contents of a startup configuration file, e.g., the /etc/profile file or the .login or .

bashrc script within the home directory of the account. Environment variables are by no means set in stone and can be altered by the user in most cases.

Table 13.11. Common Environment Variables

HOME	the path to the home directory
IFS	Internal Field Separator characters. Usually space, tab and newline
PATH	colon separated list of directories to search for the command to execute
PS1, PS2	The strings to use as the primary and secondary prompts
PWD	The current working directory
SHELL	absolute path to the executable that is being run as the current shell session.
USER	The name of the user that runs the current session

The scope of variables within a shell is global (in **bash**, you can use the local keyword to define local variables within functions). However, there is a catch: subshells - new shells, spawned by the current shell - do not automatically inherit the variables of the calling shell (script). To enable inheritance of a variable, you need to use the export keyword. When a variable is exported its data is accessible by its child-processes. Such a variable is called an "*environment variable*". See the example captured below:

```
$ cat test
#!/bin/sh
echo ${a}
exit0
$ a=4
$ ./test

$ export a
$ a=4
$ ./test
4
$ _
```

Also, a subshell is *not* able to change the content of the variable in the calling shell - in other words: the subshell uses a *copy* of the variable. By using parentheses, you can force a subshell to be started. See this example:

```
#!/bin/sh
export a=4
```

```
echo $a
( a=9; echo $a )
echo $a
exit 0
```

This will produce the output:

```
4
9
4
```

Special variables

Within a shell script, a number of special variables are available. They are filled by the shell on execution and cannot be influenced (set) by the user; therefore we will call them "environment pseudo-variables" from now on. The table below lists most of them:

Table 13.12. Common Environment Pseudo Variables

$0	The full name of the current script.
$1 .. $9	The first to ninth argument passed to the shell
$#	The number of arguments passed to the shell or the number of parameters set by executing the set command
$*	The values of all the positional parameters as a single value
$@	Same as $*, except when double quoted it has the effect of double-quoting each parameter
$$	The PID of the current shell
$!	The PID of the last program sent to the background
$?	The exit status of the last command not executed in the background
$-	lists the current options, as specified upon invocation of the shell (some may be set by default).

As you can see, there are "only" nine environment pseudo variables that represent parameters. What, if you have more than nine parameters? You could use the $@ or $* variable and use a loop (the section called "Branching and looping") to list all variables - like this:

```
#!/bin/sh
for parameter in "$@"
```

```
do
        echo $parameter
done
exit 0
```

This loop will set the variable parameter to the next argument and display it. Assuming you saved this script as partest, the following session could be run:

```
$ sh partest 1 2 3 4 5 6 "seven is a nice number" 8 9 10 11 12
1
2
3
4
5
6
seven is a nice number
8
9
10
11
12
$ _
```

As an experiment, you might want to replace the $@ in the partest script with $* and rerun the script using the same arguments:

```
$ sh partest 1 2 3 4 5 6 "seven is a nice number" 8 9 10 11 12
1 2 3 4 5 6 seven is a nice number 8 9 10 11 12
$ _
```

Now remove the quotes, and you would get:

```
$ sh partest 1 2 3 4 5 6 "seven is a nice number" 8 9 10 11 12
1
2
3
4
5
6
seven
is
a
nice
```

```
number
8
9
10
11
12
$_
```

You might experiment a bit further by adding a line that also prints the contents of the $# pseudo-variable. Another method to iterate through more than 9 arguments is by using the special shell keyword shift. An example that uses another form of looping (the section called "Branching and looping"), the while loop, illustrates the shift:

```
#!/bin/sh
while [ ! -z "$1" ]
do
      echo "{" $1 "}"
      shift
done
exit 0
```

This program will cycle through all of its parameters and display them, one by one. Here is yet another method: use two pseudo-variables and shift:

```
#!/bin/sh
while [ $# -ne 0 ]
do
      echo "$1"
      shift
done
exit 0
```

Branching and looping

It is often necessary to conditionally execute parts of your scripts. Additionally, a number of methods is implemented which allow you to perform controlled iteration of blocks of code (looping).

The shell supports many branching options. The if is used most often. It enables you to conditionally execute blocks of code. The condition should immediately follow the if statement. Next comes the block of code to execute, delimited by the keywords then and fi:

```
if {condition}
```

```
then
	{code}
	{more code}
fi
```

The condition should evaluate to a value of zero ("*true*") or one ("*false*"). The block of code will be executed if the condition is *true*. Optionally, you can use the else keyword to provide a container for a block of code that will be executed if the condition evaluates to *false*:

```
if {condition}
then
	{code executed if condition evaluates true}
else
	{code executed if condition evaluates false}
fi
```

.. or, to be concrete:

```
if test 100 -lt 1
then
	echo "100 is less than 1 - fascinating.."
else
	echo "1 is less than 100. So, what else is new?"
fi
```

We made use of the shell built-in command **test** in this example. **test** evaluates logical expressions and is often used to form the condition for looping and branching instructions. test takes boolean statements as its argument and returns a value of zero or one. The shortened form of the **test** command is formed by enclosing an expression in square brackets [], like this:

```
if [ 100 -lt 1 ]
then
	echo "100 is less than 1 - fascinating.."
else
	echo "1 is less than 100. So, what else is new?"
fi
```

Be certain to include white space between the square brackets and the statement. The **test** command is often used in conjunction with the shell's boolean operators. This relies on the shell's behavior to evaluate just enough parts of an expressions to ensure its outcome:

```
[ $YOUR_INPUT = "booh" ] && echo "Are you a cow?"
```

The part of the expression to the right of the double ampersands (boolean AND) will never evaluate if the part to the left evaluates to false. The **else** keyword can be followed by another **if** keyword, which allows nesting:

```
if [ 100 -lt 1 ]
then
      echo "100 is less than 1. Fascinating.."
else
    if [ 2 -gt 1 ]
    then
      echo "2 is bigger than 1. Figures."
    else
      echo "1 is bigger than 2. Fascinating.."
    fi
fi
```

The examples given so far are, of course, not very realistic, but serve the purpose of clarifying the concepts. In practice, testing is mostly done against variables, as demonstrated in the following example (which demonstrates a shorter form of the **else ... if** statements: the **elif** statement. It allows sequential matching of conditions without the need to nest **if.. then.. else .. fi** constructs):

```
#!/bin/sh

VAR=2

if [ $VAR -eq 1 ]
then
    echo "One."
elif [ $VAR -eq 2 ]
then
   echo "Two."
else
   echo "Not 1, not 2, but $VAR"
fi
```

Another branching mechanism is the **case** construct. The syntax for this construct is:

```
case {variable} in:

    {expression-1}) {code} ;;

    {expression-2}) {code} ;;
```

```
esac
```

The **case** construct is used to test the contents of a variable against a regular expression (as described in the section called "Regular Expressions"). If the expression matches, a corresponding block of code will be executed. That block starts after the closing parenthesis and should be terminated using a sequence of two semi-colons (;;). If a matching expression is found, the script will continue just after the esac statement. An example:

```
Language="Cobol"

case ${Language} in

    Java|java) echo "Java";;
    BASIC)     echo "Basic";;
    C)         echo "Ah, a master..";;
    *)         echo "Another language";;
esac
```

Note the use of the the wild card "*)", which matches any pattern. If none of the other expressions match the variable, the script prints a default message.

Loops iterate through a block of code until or while some condition is met. Conditions might be, for example, that a list of values is exhausted, a command returned a true or false value or a given variable or set of variables reached a value. Within a POSIX shell there are three types of loops: the for, while and until constructs. Previously, we gave you two examples of looping: the for and the while loop. A few additional examples will, hopefully, clarify their proper use:

for loops. the syntax for a **for** loop is

```
for {variablename} in {list-of-values}
do
   {command} ...
done
```

{list-of-values} is a whitespace separated list of values. This type of loop fills the variable named in {variablename} with the first element of the list of values and iterates through the command(s). It keeps iterating until all values of the list have been processed, or the loop is terminated otherwise, e.g., using the **break** command. For example:

```
$ cat list.sh
#!/bin/sh

# this loop lists some numbers
```

```
#
for VALUE in 19 2 1 3 4 9 2 1 9 11 14 33
do
	echo -n "$VALUE "
done
echo
exit 0
$ sh list.sh
19 2 1 3 4 9 2 1 9 11 14 33
$ _
```

It is possible to use the output of a command to generate a list of variables, and this can be combined with elements you define yourself, as is demonstrated in this capture of a session:

```
$ cat ./count
#!/bin/sh

# this loop counts from 1 to 10
#
for VALUE in zero `seq 3`
do
	echo $VALUE
done
exit 0
$ sh count
zero
1
2
3
$ _
```

A `for` loop expects its parameters to be literal strings. Hence, execution of the **seq** program is forced by putting its name and parameters between backticks. Otherwise, the loop would just iterate the loop using the strings "`zero`", "`seq`" and "`3`" to fill the `VALUE` variable.

Any loop can be terminated prematurely using the **break** statement. Execution of the code will be resumed on the first line after the loop, for example, this code snippet will print just "`A`" (and not "`B`" and "`C`"):

```
for var in A B C
do
	break
	echo "this will never be echoed"
```

```
done
echo $var
```

If, for some reason, the remaining part of the code within a loop needs not be executed, you can use the **continue** statement.

```
for var in A B C
do
	echo $var
	continue
	echo "this will never be echoed"
done
```

The code snippet above will print

```
A
B
C
```

while loops. If you want to execute code *while* some condition holds true, you can use the while statement. A while function executes statements within a corresponding do .. done block, e.g.:

```
while true
do
	echo "this will never be echoed"
done
echo "this is echoed"
```

In our example, we used the common program **true**, which will just exit with an exit code 0 (zero). Its counterpart, **false** will just exit with the value 1. Note that a while loop expects a list, much as the for loop does. The while loop, however, expects a list of executable *commands* rather than a list of variables. As long as the last command in the list returns zero, the commands delimited by the do .. done keywords will be executed:

```
while
	false	# returns 1
	echo "before"  # returns 0 -> code within do..done block will be executed
do
	break
	echo "this will never be echoed"
done
echo "this is echoed"
```

Execution of this code would result into the following output:

```
before
this is echoed
```

Note, however, that you almost never see more then one command within the list. Often, this will be the **test** program. As you see, this loop-type can also be terminated prematurely using the **break** command.

```
while true
do
	break
	echo "this will never be echoed"
done
echo "this is echoed"
```

You can force execution of the next iteration by using the **continue** command:

```
stop=0
while test $stop -ne 1
do
	stop=1
	continue
	echo "this will never be echoed"
done
echo "this is echoed"
```

The example above also demonstrates the aforementioned usage of the program called "**test**". Remember: **test** validates the expression formed by its arguments and exits with a value of *true* (0) or *false* (1) accordingly, see the manual page for **test(1)**.

> **Note**
>
> The **bash** shell has a built-in version of **test**. It is described in the manual page of bash.

until loops. The until function is very similar to the while function. It too expects a list of executable *commands*, but the corresponding block of code - again delimited by the do .. done statements - will be executed as long as the last command in the list returns a *non-zero* value:

```
until true
do
```

```
        echo "this will never be echoed"
done
echo "this is echoed"
```

The following example lists the values from 1 to 10:

```
A=0
until [ $A -eq 11 ]
do
        A=`expr $A + 1`
        echo $A
done
```

Note the method used to increment the value of a variable, using the execution (backticks) of the **expr** tool.

Functions

Functions are just a set of shell commands grouped together under a unique name. Once a function has been defined, you can call it using its name as if it was a standalone program. Shell functions allow programmers to organize actions into logical blocks. Instead of having to repeat the same lines over and over again, which would result in long scripts, performance penalties and a far bigger chance of erroneous code, similar lines of code can be grouped together and called by name. This stimulates modular thinking on behalf of the programmer and provides reuse of code. It is also easier to follow the script's flow.

A function is defined by declaring its name, followed by a set of empty parentheses () and a body, denoted by curly braces {}. Within that body the commands are grouped, e.g.:

```
#!/bin/sh

dprint() {
        if [ $DEBUG ]
        then
                echo "Debug: $*"
        fi
}

dprint "at begin of main code"

exit 0
```

Save this script in a file named td. Try running it. It will output nothing. Now set and export

the DEBUG variable and run it again:

```
$ export DEBUG=1
$ ./td
Debug: at begin of main code
$ _
```

The if ... then construct used within the function will be explained later on. A number of things can be observed here. First of all, the function needs to be declared *before* it can be called. Also, the method to call a function is demonstrated: simply use its name. As you see, you can specify parameters. Note that the definition of the function does not require specification of the number of parameters. When calling a function you are free to specify as many parameters as you would like. You will need to write code within the function to ensure that the proper number of parameters is passed. The $* pseudo-variable can be used within the function to list (if any) the argument(s). Indeed, pseudo-variables $1..$9, $#, $@ and $* are all available within the function as well, and differ from the pseudo-variables with the same name within the script. To demonstrate this, save the following program in a file named "td2":

```
#!/bin/sh
f1 () {
        echo "in function   at=" "$@"
        echo "in function hash=" "$#"
        echo "in function star=" "$*"
}

echo "in main script at=" "$@"
echo "in main script hash=" "$#"
echo "in main script star=" "$*"

f1 one flew over the cuckoos nest

exit 0
```

Running it, with the specified command, will have the following result:

```
$ sh ./td2 coo coo
in main script at= coo coo
in main script hash= 2
in main script star= coo coo
in function   at= one flew over the cuckoos nest
in function hash= 6
in function star= one flew over the cuckoos nest
$ _
```

Though functions do not share the set of pseudo-variables with the rest of the script, they *do* share all other variables with the main program. If a function changes the content of a variable or defines a new one, its contents will be available in the main script as well. The following script clarifies this:

```
#!/bin/sh

SET_A () {
      A="new"
      B="bee"
}

A="old"

echo $A       # will display "old"

SET_A         # sets variable in script

echo $A       # will display "new" now
echo $B       # and this will display "bee"

exit 0
```

On most modern POSIX-compliant shells they will work just fine, to avoid confusion, it is best to refrain from using variables and functions with the same name in one script.

You are allowed to use recursion - in other words: you are allowed to call a function *within itself*. However, it is important to provide some sort of escape, otherwise your script will never terminate:

```
#!/bin/sh

call_me_4_ever() {

      echo "hello!"
      call_me_4_ever
}

call_me_4_ever
exit 0
```

This script will run forever, happily printing out greetings. A more usable example follows.

Please study it carefully - it demonstrates various concepts we will explain later on.

```
/----------------------------------------------------------------------
01 | #!/bin/sh
02 |
03 | seqpes() {
04 |
05 |     fail()
06 |     {
07 |         echo failed: $*
08 |         exit 1
09 |     }
10 |
11 |     [ $# -eq 3 ] || fail argument error
12 |
13 |     local z=`expr $1 + $3`
14 |
15 |     echo $1
16 |     [ $z -le $2 ] && seqpes $z $2 $3
17 |     echo $1
18 | }
19 |
20 | seqpes $1 $2 $3
21 |
22 | exit 0
/----------------------------------------------------------------------
```

Can you figure out what this program does? Probably not. Many people need to run this first to understand its function.

The example illustrates the concept of recursion, nested functions, local variables and alternate testing syntax. For those of you who look at this script with awe, it may come as a surprise to learn that this is a very typical example of *illegible code*. You are not supposed to write code like this (see the section called "Some words on coding style" for an explanation why not).

That said, lets look at the improved version:

```
/----------------------------------------------------------------------
01 | #!/bin/sh
02 |
03 | # This script prints sequences of values, counting up first,
04 | # then down again. It needs 3 parameters: the start value,
```

```
05 | # the maximal value and the increment. It uses recursion
06 | # to achieve this. See the comment before the function
07 | # seqpes() found below for an explanation.
08 |
09 | # fail() prints its arguments and exits the script with an
10 | # error code (1) to signal to the calling program that
11 | # something went wrong.
12 | #
13 | fail()
14 | {
15 |     echo failed: $*
16 |     exit 1
17 | }
18 |
19 |
20 | # seqpes() needs 3 arguments: the current value, the maximal
21 | # value and the increment to use. It will print the current
22 | # value, increment it with the increment to use and then will
23 | # create a new instance of itself, which acts likewise,
24 | # until the printed result is equal or bigger than the maximal
25 | # value. Each time the function is called by itself, a new set
26 | # of pseudo-variables ($1..$3) is created and kept in memory.
27 | # When the maximal value finally has been reached, the last
28 | # function exits and gives control back to the function that
29 | # called it, which acts likewise until the calling
30 | # program finally regains control. This ASCII art tries to clarify
31 | # this:
32 | #
33 | #  seqpez 1 3 1
34 | #  print $1 ..... (1)
35 | #  |  seqpez 2 3 1
36 | #  |  print $1 ..... (2)
37 | #  |  |  seqpez 3 3 1
38 | #  |  |  print $1 ..... (3)
39 | #  |  |  print $1 ..... (3)
40 | #  |  |  $1=3
41 | #  |  print $1 ..... (2)
42 | #  |  <--exit
43 | #  print $1 ..... (1)
44 | #  <--exit
45 | #
46 | seqpes() {
47 |
```

```
48 |    # print the current value and calculate the next value
49 |    #
50 |    echo $1
51 |    next_value=`expr $1 + $3`
52 |
53 |    # check if we already surpassed the maximal value
54 |    #
55 |    if [ $next_value -le $2 ]
56 |    then
57 |   # we need another iteration.
58 |       seqpes $next_value $2 $3
59 |    fi
60 |
61 |    # Done. Print this value once more:
62 |    echo $1
63 | }
64 |
65 | # The script starts here, and checks its parameters. Note that
66 | # the code that validates the arguments is very incomplete! We
67 | # just check for the number of arguments, not for their values,
68 | # for example. This segment definitely lacks quality.
69 | #
70 | if [ $# -ne 3 ]
71 | then
72 |    # print and exit
73 |    fail argument error
74 | else
75 |    # call recursive function
76 |    seqpes $1 $2 $3
77 | fi
78 |
79 | exit 0    # tell calling program all went well
  /--------------------------------------------------------------------
```

As you see, this code has almost four times the amount of lines our previous example had. However, it uses a more generic syntax, has many lines of comments and does not use local variables and nested functions. It works just as well as the previous example and is self-explanatory - hence, nothing more needs to be said about it.

Here documents

A so-called *here document* allows you to use a series of text-lines directly from a shell script as input for a command. The following example will print a help-text:

```
/-----------------------------------------------------------------
01 |#!/bin/sh
02 |
03 |usage()
04 |{
05 |   cat << ENDCAT
06 |   Use $0 as follows:
06 |      $0 spec
08 |   where spec is one of the following:
09 |      red    - to select red output
10 |      green  - to select green output
11 |      blue   - to select blue output
12 |ENDCAT
13 |
14 |   exit 1
15 |}
16 |
17 |test -z "$1" && usage
18 |
19 |.... the rest of the script ...
/-----------------------------------------------------------------
```

This program will check in line 17 if the first argument is present (a way to test if any arguments were given at all). If not, the program will call a function called usage. Inside usage a *here-document* is used to feed the usage text to **cat**. The *here-document* starts in line 05 and ends in line 12. It consists of:

- << (line 05) followed by:
- a *label* definition (ENDCAT here, line 05).
- a closing label: ENDCAT in line 12. The closing label **must** start at the left (as shown) and must be spelled exactly like the label definition (in line 05).

The text between the begin and end of the *here document* (that is text in lines 06 until 11) will be passed to **cat**, which will display it. The output of the usage function (supposing that the script is called mysetcolor) will be:

```
Use mysetcolor as follows:
   mysetcolor spec
where spec is one of the following:
   red    - to select red output
   green  - to select green output
   blue   - to select blue output
```

Note

Here-documents can be controlled in several ways. Finding these out is left as an exercise to the reader.

Advanced topics

By now you should have learned enough to pass the LPIC-2 exam. However, shell programming is an art in itself. It goes well beyond the scope of this book to teach you everything there is to know about it. On a number of topics, if you would like to read further, please check the bibliography. Amongst the topics perhaps warranting further study are:

- how to include external scripts in your own (dotting in scripts)
- how to use arithmetic within your script
- how to trap signals sent to your script
- how to use in-line redirection
- setting parameters using "set"
- how to use the **eval/exec** commands

Debugging scripts

To trace a script's execution, you can use the command **set**. The option -x causes the shell to print each command as it is evaluated. Setting this option also enables a scripter to see the results of variable substitution and file name expansion. The -v option instructs the shell to print a command *before* it is evaluated. Using both parameters enables you to study the inner workings of the script, however it results in a rather elaborate and sometimes confusing output. To switch off debugging, use the "+x" option.

Experienced shell programmers often use debug functions and statements to debug a script. For example:

```
#!/bin/sh

DEBUG=1  # remove this line to suppress debugging output

debug()
{
    [ $DEBUG ] && echo DEBUG: $*
}


WORLD="World"
```

```
debug "WORLD = $WORLD"
echo "Hello World"
debug "End"
```

By setting the DEBUG variable, you can control whether or not debugging output will be sent. If you are done debugging, you can either remove the "debug" lines and function or leave them alone and remove the line that sets DEBUG.

You can use your own set of debugging functions to temporarily stop or delay execution of your script. By using the power that Unix gives you, you can even write a very basic debugger that enables you to set breakpoints and interactively view the environment. See the example below:

```
#!/bin/sh

DEBUG=1  # remove this line to suppress debugging output

debug_print()
{
    [ -z "$DEBUG" ] || echo DEBUG: $*
}

debug_breakpoint()
{
    while [ ! -z "$DEBUG" ]
    do
      echo "DEBUG: HELD EXECUTION. (q)uit, (e)nvironment or (c)ontinue?"
      stty raw
      stty -echo
      A=`dd if=/dev/tty count=1 ibs=1 2>/dev/null`
      stty -raw
      stty echo
      case $A in
          q|Q) exit;;
          e|E) set;;
          c|C) break;
      esac
    done
}

debug_delay()
```

```
{
    [ -z "$DEBUG" ] || sleep $1
}


WORLD="World"
debug_print "WORLD = $WORLD"
echo "Hello World"
debug_print "End"
while true
do
    echo "loop.."
    debug_breakpoint
    debug_delay 1
done
```

This script demonstrates a number of techniques - we leave it up to you to study them as an exercise. Note, that the techniques used above will only work for interactive scripts - that is: scripts that have a controlling tty. Try running this script, both with and without the DEBUG flag.

Some words on coding style

The power of the Unix programming environment tends to seduce a lot of programmers into writing unreadable code. Often, the argumentation is that terse code will be more efficient. In practice, this is often not true. Additionally, many novice programmers tend to try to "show off" their knowledge. Many scripts are written that are barely readable and have no indentation. Bad code uses shortcuts, nesting and recursion whenever possible and uses uncommon constructs. All this is then spangled with esoteric regular expressions. *You must never write code like this. Remember: you are almost a certified professional. The main reason you will be hired is not to impress your peers, but to enable others to do their work - including your peers.*

Good programmers are consistent in how they write. They judge code readability to be more important than (minor) performance gains. They refrain from things like nested functions or functions that bear the same name as a variable. They user proper indentation and test their programs. They also test if their newly-written program really works. execute, e.g. For example:

```
.. code ..

cd $WORKDIR
if [ $? -ne 0 ]
then
```

```
    echo could not change to $WORKDIR
    exit 1
fi

.. more code ..
```

A good programmer spends a lot of time documenting his work, both inside the code and in separate documentation. He is aware that some of his peers may lack his skills and therefore tries to use the simplest approach to solve a problem. He uses plain English to clearly and succinctly explain how his code works. He refrains from snide remarks, the usage of weird acronyms and emoticons. In general: *be brief, but complete*.

Prev | Up | Next
Using Perl | Home | Using sed

Using sed

Revision: $Revision: 1.3 $ ($Date: 2004/01/30 10:22:06 $)

sed is an abbreviation of *stream editor*. It can edit information on the fly while reading information from either *standard input* or one or more files.

As shown in the section called "Regular Expressions", **sed** operates with the classic Regular Expressions. The Regular Expressions described here are valid for GNU sed (the examples are tested with GNU sed version 3.02). Note that very old, non-POSIX compliant versions of **sed** may not have the extra multipliers and grouping support.

Behaviour

Note that **sed** will pass all information it reads on to *standard output* by default.

Since we can control information when it is processed by **sed**, we can delete part of the information before it reaches *standard output* or we can change part of the information so that what comes out on *standard output* is different from what was read.

But, if we do not intervene, information is shown as-is by **sed**.

There is also the possibility of *reverting* this behavior. When the -n option is specified, **sed** will only show information when explicitly asked to do so.

Calling sed

There are two ways to call *sed*. In the first form, **sed** reads from *standard input*:

```
... | sed [flags] 'sed expression'
```

In the second form, **sed** will read information from one or more files:

```
sed [flags] 'sed expression' file(s)
```

The crucial part in both forms is the *sed expression*. The *sed expression* specifies what will be done to the information. What it looks like will be discussed further on. How a *sed expression* is specified is shown here.

Note

The *sed expression* may contain spaces and characters that are subject to shell expansion. To prevent expansion by the shell, the complete *sed expression* must be embedded in single forward quotes, as shown above. **sed** will not see these quotes.

When multiple *sed expression*s are desired, each must have a -e flag before it. The first form:

```
... | sed -e 'sed expr1' -e 'sed expr2'
```

The second form:

```
sed -e 'sed expr1' -e 'sed expr2' file(s)
```

The *sed expression*

The *sed expression* allows you to control the information passing through **sed**. It can, however, have many different forms.

The general form of a *sed expression* is

addresssedcommand

That is: an *address* immediately followed by a *sedcommand*. If *address* is valid for a line of information, then *sedcommand* will be applied.

An *address* can be a Regular Expression (between slashes), or it can be a line-number specification. Some *sedcommand*s may need extra additions.

Regular Expression specification. A form with a Regular Expression specification as *address* looks like:

/regex/sedcommand

An example of this is:

```
/=/d
```

The Regular Expression specification is /=/, the *sedcommand* is d. Together, these tell **sed** not to display lines that contain an = character.

Line number specification. The *address* can be a line-number specification (abbreviated as *linespec*):

linespecsedcommand

(that is: *linespec* immediately followed by *sedcommand*). The *linespec* is a specification of line number(s). They can be specified as a single number or as a *range*: two numbers with a comma between them. An example of this is:

```
1,3d
```

The result of this is that the first three lines read by **sed** will not be shown. The other lines will, of course, be shown.

The most frequently used *sedcommand*s

The *delete* and *substitute* commands are probably the most frequently used commands.

The delete command

The general form is

*address*d

That is: *address* immediately followed by a d. Using this, **sed** will omit the information that matches *address* from the information that is being processed by **sed**. Information **not** matching *address* **will**, of course, be passed on.

An example of this is:

```
sed '1d' /etc/passwd
```

This command line[22] will result in all lines of /etc/passwd, except the first one, to *standard output* (the screen in this case).

This is another example, using a *range*:

```
sed '1,10d' myfile
```

This will cause the contents of myfile to be shown, minus the first ten lines.

Using a Regular Expression as *address* works the same way:

```
sed '/^$/d' myscript | less
```

This will pass all non-empty lines from myscript to the **less** command.

A little extension:

```
sed '/^ *$/d' myscript > newscript
```

Here the result will be that all lines in myscript, except those that contain nothing or only space (s), will be written to newscript.

The substitute command

With the *substitute* command, you can change some of the information before it is sent to the *standard output*. The other information will go there as well, but unchanged.

The most general form of the substitute *sedcommand* is:

*address*s/*regex*/*replacementstring*/*modifiers*

The *address* works as a line selector, as shown earlier. The *address* part can be omitted, in which case the substitute is attempted on all lines. The *modifiers* can also be omitted, in which case the defaults apply (more on this later). Since both the *address* and the *modifiers* can be omitted, a simplified command would look like this:

s/*regex*/*replacementstring*/

This format, of course, is very famous.

The *substitute* works as follows: the part of the line on which *regex* fits is replaced by *replacementstring*.

sed reads its information line by line, as do many other Unix commands. A *substitute* works only *once* per line: the first location (seen from the left) where the Regular Expression matches is replaced by the *replacementstring*.

This behavior can be changed by the g modifier. When the g modifier is specified, each part of the input string where the Regular Expression fits is replaced by the replacement string. The g (global) means: as many times as possible within the input line.

Now, let's look at this in practice. Suppose the following information is generated by **echo**:

```
echo http://www.x.z/HTML Files/Pag 1.html | sed 's/ /%20/'
```

In this example, the input string contains two spaces. The **sed** command-line replaces one space (the first) by %20. So the result will be:

```
http://www.x.z/HTML%20Files/Pag 1.html
```

That is, the first space is replaced by %20, the second is not.

> **Note**
>
> What is shown here corresponds to the so-called *urlencoding*. But this is not completely covered here. At least, percent signs in the input should be replaced first.

To replace all spaces, the g modifier needs to be specified:

```
echo http://www.x.z/HTML Files/Pag 1.html | sed 's/ /%20/g'
```

Now the result is:

```
http://www.x.z/HTML%20Files/Pag%201.html
```

That is, **both** spaces are replaced by %20.

Suppose you've got a file containing lines with URLs as well as lines containing other information. You want to replace spaces in the URLs, **but only in the URLs**. In this case, an *address* can be used to select the URLs:

```
/^http:/s/ /%20/g
```

Suppose file mystuff contains the following:

```
http://a.b.c/Item 1/index Main.html
some other information on a separate line
http://a.b.c/Item 2/index Subpage.html
```

This file can be processed as follows:

```
sed '/^http:/s/ /%20/g' mystuff
```

The output will be:

```
http://a.b.c/Item%201/index%20Main.html
some other information on a separate line
http://a.b.c/Item%202/index%20Subpage.html
```

Outputting changed lines only

Recall that the -n flag reverses the behavior of **sed**: information read is not passed on by default. To show information some action must be taken. The substitute command has a p modifier that does exactly that: outputting those lines that are changed. Using both the -n flag and the p modifier together will output only those lines that are changed.

sed -n 's/*regex*/*replacement*/p' *infile(s)*

Another example is this:

```
sed -n 's/\b[Ll]inux\b/LINUX/p' txtfile
```

This will read all lines from txtfile, but show only lines that contain either linux or Linux with both linux and Linux replaced by LINUX.

The &: the matching part

The precise part of an input line on which the Regular Expression matches is represented by &, which can then be used in the replacement part. An example is (split across lines for readability):

```
echo Now is 15 Feb 2002 | \
    sed 's/[0-9]\{1,2\} \+[A-Za-z][a-z]\+/=&=/'
```

This will replace the date part with the date surrounded by = characters. The part of the input that does not match the Regular Expression is not replaced. Instead it is shown literally. So the output is:

```
Now is =15 Feb= 2002
```

Grouping in sed

Grouping can be used in **sed**, at least in the GNU version that is used by default in Linux. A group is opened with "\(" (a backslash and an opening parenthesis) and closed with "\)" (a backslash and a closing parenthesis). Grouping can be used in combination with *back-referencing*.

Back-references

Remember that a *back-reference* is the re-use of a part of a Regular Expression selected by *grouping*.

Back-references in **sed** can be used in both a Regular Expression and in the *replacement* part

of the *substitute* command.

For example, the following will **not** show lines in files f1 and f2 that contain four identical uppercase-letters:

```
sed '/\([A-Z]\)\1\1\1/d' f1 f2
```

The following *substitute* will replace each series of four identical uppercase-letters by four X characters:

```
ls -la | sed 's/\([A-Z]\)\1\1\1/XXXX/g'
```

The following *sedexpression* places any uppercase-letter between square brackets:

```
s/\([A-Z]\)/[\1]/g
```

Remember that the *replacement* part is not a Regular Expression, so the square brackets are literal characters.

For example:

```
echo Hello There | sed 's/\([A-Z]\)/[\1]/g'
```

The output of the above command line will be:

```
[H]ello [T]here
```

Replacing the whole input

Remember that a **sed** *substitute* command will replace only the part on which the Regular Expression matches. Parts of the line that do not match are sent unchanged to the *standard output*.

When you **do** want to match the whole line, the following must be true:

1. the Regular Expression must match the complete line
2. the desired part must be selected using grouping

Let's look again at the date example discussed earlier (split across lines for readability):

```
echo Now is 15 Feb 2002 | \
  sed 's/^.*\b\([0-9]\{1,2\} \+[A-Za-z][a-z]\+\).*$/\1/'
```

The output is exactly:

```
15 Feb
```

Do not forget the \b word-boundary anchor: without it, the .* will match too many characters.

White space

Remember that **sed** uses the classical Regular Expressions, in which there is no easy way to specify a tabulation character.

In modern, GNU versions of **sed**, POSIX character-classes are accepted, so [[:blank:]] and [[:space:]] can be used to match white space. However, since POSIX support (required by some governments) is not yet present on all non-Linux systems, the use of POSIX character-classes may be non-portable.

Advanced sed

Lots of things can be done with **sed**. Among **sed**'s facilities are *command-grouping* and *pattern-space*. Things might grow too complex to use on the command line, so this section will show how to use a separate file with **sed** commands.

Putting *sedcommand*s in a file

It is is possible to store individual *sed expression*s in a so-called "sed script-file". Each line should contain exactly one *sed expression*. There should be no quotes around the *sed expression*s in the *sedfile*:

```
s/red/green/
/blue/d
```

The sed script-file can be called by **sed** using the -f flag. The first form:

```
... | sed -f sedfile
```

The second form:

```
sed -f sedfile file(s)
```

Other flags can be added (e.g. -n) **before** the -f flag or **after** *sedfile*. The -f and *sedfile* should be used together. Do not put other parameters between -f and *sedfile*.

Consider for example, the following **sed** command:

```
sed -e 's/<title>/title(/' -e 's/<\/title>/)/' f1
```

This can be put in a special sed command-file. We call it chgtitle.sed:

```
s/<title>/title(/
s/<\/title>/)/
```

Now all we have to do is:

```
sed -f chgtitle.sed f1
```

This is the contents of file f1:

```
<title>Everything about sed</title>
```

The output of the **sed** command line will be:

```
title(Everything about sed)
```

Using a separate *sedfile* allows you to do more complex things. Among these are *command grouping* and the *pattern buffer*.

Command grouping

sed *commands*, such as d or s, can be grouped. This can be explained using an example task: an ASCII art table that should be converted into html.

Suppose, the file orgtable contains the following table in ascii art:

```
Table 1
 20-3  3-4 10-4 17-4  1-5  8-5 15-5 22-5  5-6 19-6  pct
-----------------------------------------------------
|   |+5  |   |   |-9  |-8  |-6  |+2b |+1  |+8  | 57%|
|-2b |+8b |-6b |-9  |+1b |+7  |+8b |+1c |+6  |-7  | 60%|
|+7b |-7  |-6  |+2c |+10c|+1  |   |-9  |   |+3b | 63%|
```

We want to convert the lines inside the table into html. The first line should look like this after conversion:

```
<TR><TD>&amp;nbsp;&amp;nbsp;</TD><TD>+5</TD>
<TD>&amp;nbsp;&amp;nbsp;</TD><TD>&amp;nbsp;&amp;nbsp;</TD>
<TD>-9</TD> <TD>-8</TD> <TD>-6</TD> <TD>+2b</TD> <TD>+1</TD> <TD>+8</TD>
<TD>57%</TD> </TR>
```

To accomplish this, we have to make several substitutions, among them the substitution of white space into " " (the non-breakable-space character in HTML. We only want to do this in the table entries (that is: all lines that start with a |), not in the table header. The following commands in the *sedfile* called htmlise.sed will replace only those lines that start with a |:

```
/^|/s/^|/<TR><TD>/
/^|/s/|$/<\/TD><\/TR>/
/^|/s/|/<\/TD><TD>/g
/^|/s/ / /g
/^---/d
```

As a bonus it gets rid of the dashed line.

> **Note**
>
> The literal slashes in the replacement part need to be prefixed by a backslash to prevent **sed** from mixing it up with the slash separator.

Here is the complete call to convert the contents of orgtable to html using htmlise.sed:

```
sed -f htmlise.sed orgtable
```

There is another way to write the same thing. As can be seen, the *address* /^|/ is used four times. Commands can be grouped by using the same *address* followed by the same commands inside curly braces. This is a so-called *command group*:

```
/^|/  {
   s/^|/<TR><TD>/
   s/|$/<\/TD><\/TR>/
   s/|/<TD><TD>/g
   s/ / /g
}
/^---/d
```

We begin by matching every input line with the /^|/ pattern. If this matches, all the commands between the curly braces are applied to this line. The first and the last vertical bars are exceptions because they need only "<TR><TD>" (row and cell open in HTML) and "</TD></TR>" (cell and row close in HTML), so we substitute them first. Next we substitute each remaining vertical bar with "</TD><TD>" pattern. Last, we substitute each space by " ".

> **Note**

White space at the start of a command is allowed, so you can indent your script to reflect the grouping of the commands. You cannot put whitespace *after* a command: the end of a command must be either a newline or a semicolon. The semicolon is not part of the POSIX standard, but is supported in the GNU version of **sed**.

The pattern buffer

Suppose you want to match a fixed group of words, such as a name, but they are not on the same line. For example, in a manual you want to change the words "local unix guru" into "Local Expert". But, the word "local" could be on one line and the words "unix guru" on the next. This would be hard to match with the commands discussed so far. Enter the *pattern buffer*.

The pattern buffer is the place where the current line is stored (that is: the buffer to which the Regular Expression is matched). This is the buffer on which the commands are executed and which is printed after the last command is finished. Lines can be added to this buffer by using the N command. When the new line is appended to the former, a *newline* character is inserted between them.

It is now possible to match input like this:

```
Yesterday, our local
unix guru explained that ...
```

In this case, we have to match every combination in which the words can be divided over two lines (three possibilities). Even better: we can first remove the *newline* character. Our *sedfile* would look like this:

```
/local/ {
   N
   s/ *\n/ /
   s/local unix guru/Local Expert/
}
```

The output will be:

```
Yesterday, our Local Expert explained that ...
```

There is still a small problem though. When you run this script on the following input, the result might be not quite what you expect. Here is the input:

```
When all else fails, ask your local unix guru.

   This is an extra indented line.
```

```
This story was brought to you by:
your local unix guru
```

The result is:

```
When all else fails, ask your Local Expert.
   This is an extra indented line.

This story was brought to you by:
your local unix guru
```

First, the empty line before This is an extra ... has been removed. This is probably not what we intended. Second, the last line is unchanged (in some non-GNU versions of **sed**, the last line may disappear because **sed** crashes).

The problem is that the pattern we looked for (i.e., local) was on one line. So we did not have to read the second line before making the substitution. Because we did so anyway the newline was removed and the empty line was added to the line that contained the match. The second time this happened, the Regular Expression of the substitute command did not find a second line, so the last line of input is not changed at all.

To fix this we could add a rule to match the pattern when it appears completely on one line. This gives us the following script

```
s/local unix guru/Local Expert/
/local/ {
   N
   s/ *\n/ /
   s/local unix guru/Local Expert/
}
```

The output now becomes:

```
When all else fails, ask your Local Expert.

   This is an extra indented line.

This story was brought to you by:
your Local Expert
```

Note that this is the only way to remove newlines from a file with **sed**. Because it operates on one line at a time, it normally ignores the newline. So you can't match it, unless you add the next line to the pattern buffer. When you do that, the newline is reinserted, and you can

match it using "\n".

A stand-alone sed script

Let's go back to the htmlise.sed file used earlier. If you are getting bored of typing **sed -f htmlise.sed**, consider making this a stand-alone sed script. That is a sed script-file that becomes a command of its own.

First, start by adding a line at the beginning:

```
#!/bin/sed -f
```

Of course, the location of the **sed** program may be different.

The total file will now look like this:

```
#!/bin/sed -f
/^|/  {
   s/^|/<TR><TD>/
   s/|$/<\/TD><\/TR>/
   s/|/<TD><TD>/g
   s/ / /g
}
/^---/d
```

Now rename it to htmlise, install it in a directory that is named in the PATH environment variable, give it execute permission, and you are ready to run the command everywhere:

```
htmlise orgtable
```

---

[22] The quotes are not really needed here, but added for completeness.

Using awk

Revision: $Revision: 1.12 $ ($Date: 2004/01/29 21:20:25 $)

Resources and further reading: Robbins96; **man awk**

awk is an interpreted language, used mostly to extract metadata from data files. It can be used to select rows and columns from a file and to calculate derived data, such as sums. It is also often used to filter input into a more readable format, such as log-file data.

awk uses blocks of statements, prepended with a so-called *pattern*. **awk** opens its inputfile (s), reads lines of text from them and matches the contents of the inputlines with the patterns as specified in its program. If a pattern matches, the corresponding block of code is executed. Such patterns can be regular expressions (see the section called "Regular Expressions"), but a number of other forms are supported as well.

awk can be used to extract reports from various input files, to validate data, produce indices, manage small databases and to play around with algorithms to be used in other programming languages. Some rather complex programs have been written in **awk**. However, **awk**'s capabilities are strained by tasks of such complexity. If you find yourself writing **awk** scripts of more than, say, a few hundred lines, you might consider using a different programming language, such as Perl, Scheme, C or C++.

To give you an impression of the look and feel of an **awk** program, we offer the following example. Don't worry if you do not understand (all) this code (yet), we will clarify it later.

```
function resume(line, part)
{
  len=length(line)
  if (len > ( 2 * part) )
  {
    retval = substr(line,1,part) " ... " substr(line,len-part)

  } else {

    retval = line
  }
  return retval
```

```
}

/^90:/ {
  print resume(substr($0,4),4)
}

!/^90:/ {
  print resume($0,4)
}
```

The name **awk** is an acronym and stands for the initials of its authors: Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan. It also pokes some fun at the somewhat *awkward* language that **awk** implements. The original version was written in 1977 at AT&T Bell Laboratories. In 1985, a new version made the programming language more powerful, introducing user-defined functions, multiple input streams and computed regular expressions. **awk** was enhanced and improved over many years and eventually became part of the POSIX Command Language and Utilities standard.

The GNU implementation, **gawk**, which is the version in use on most Unix systems nowadays, was written in 1986. In 1988, it was reworked to be compatible with newer (non-GNU) **awk** implementations. Current development focuses on bug fixes, performance improvements and standards compliance.

Once you are familiar with **awk**, you will most likely get into the habit of typing simple one-liners, for example, to extract and/or combine fields from input files. Such a one-liner is typically built like this:

```
$ awk 'program' input-file1 input-file2 ...
```

where program consists of a series of *patterns* (patterns) and *actions* (more about these later). This command format instructs the shell to start **awk** and use the program (in single quotes to prevent shell expansion of special characters) to process records in the input file (s).

Longer **awk** programs are often put in files. These files are either called from the command-line interface:

```
$ awk --file programname input-file1 input-file2 ...
```

or they are contained in files that start with the line:

```
#!/usr/bin/awk
```

and have the execution bit set (**chmod +x**).

Now that we have an overview, let's go in and take a closer look.

Generic flow

As stated before, an **awk** program reads lines and processes those lines, using blocks of code. A block of code is denoted by curly braces. A typical program consists of one or more of these blocks, often preceded by patterns that determine whether or not the code in the code block should be executed for the current line of input.

More formally, an **awk** program is a sequence of pattern-action statements and optional function-definitions:

```
pattern   { action statements }
```

.. and optional ..

```
function name(parameter list) { statements }
```

For each line in the input, **awk** tests to see if the record matches any pattern specified in the program. For each pattern that matches, the associated action is executed. The patterns are tested in the order they occur in the program.

Patterns can have various forms, for example, regular expressions or relational expressions; pattern types are described later (patterns).

Actions are lines of code to be executed by **awk**. The regular programming structures (branching and looping) are available; so are variables and arrays. Conditional testing, matching on regular expressions and various operators and assignments are also available. All of these components are described below. The # (hash) is used to start a comment, the remainder of the line will be ignored.

Variables and arrays

awk supports the use of variables. There are a number of *internal variables* whose values either can be read or set by both **awk** and the programmer. These can be used to influence **awk**'s behavior or to query **awk**'s environment. The programmer may also set other variables, which automatically come into existence when first used. Their values are floating point numbers or strings: automatic conversion is performed by **awk** if the context requires it.

Only one-dimensional arrays are supported. Arrays are always subscripted with strings, for example, `x[1]` (the 1 is automatically converted to the string "1") or `x[ray]`. You may also use a list of values (a comma-separated list of expressions) as an array subscript since these lists are converted to strings by **awk**. Simulations of multi-dimensional arrays can be done

this way:

```
i="X"
j="Y"
x[i,j]="text"
```

In this example, the array "x" is subscripted by a list of expressions. However, the list is converted to a string first. **awk** does this conversion by concatenating the values in i and j using a special separator (by default the hexadecimal value \034). Thus, the subscript would be the string "X\034Y", which is an acceptable subscript for **awk**'s one-dimensional associative arrays.

To delete an entire array, you can use the command:

```
delete array
```

To delete an array named scores, for example, you could issue the command delete scored. Additionally, you can delete one member of an array by using the construction

```
delete array[index]
```

So, for example, delete scores[9]. Note that this just deletes the element from the array, but does *not* rearrange the members to close the "gap".

Input files, records and fields

awk normally is used with text-file input. Within **awk** text files are seen as sets of *records* and *fields*. In most cases, a simplified view can be used: records are lines of text, delimited by a newline. However, it is possible to set your own record separator by setting the variable RS, which can be a regular expression or a single character. When RS contains a regular expression, the text in the input that matches this regular expression will separate the record.

awk splits the input records into fields. By default, (sequences of) white space are seen as a separator. It is possible to use the value of the FS variable to define the field separator. By setting FS to the null string, each character in the input stream becomes a separate field. If FS is a single character, the records are divided into fields using that single character. In all other cases, FS should be a regular expression, and the fields will be separated by the characters that match that regular expression.

To refer to a field, specify a dollar sign followed by a numerical representation of it's position within the record. To refer to the first field, for example, you would use $1, the fifth field would be $5. $0 is the whole record. It is acceptable to refer to a field using a variable, like this:

```
n = 7
print $n
```

This prints the seventh field from the input record. To determine the number of fields in the input record, you can use the variable NF. This example also demonstrates the use of the print statement, which prints its arguments to standard output, and the use of the *internal variable* NF, which contains the number of fields in the current input record. More about internal variables will be explained later on.

To split records into fields, you can also set the variable FIELDWIDTHS to a space-separated list of numbers: the records will be seen as sequences of fields with fixed width, as specified in the variable, and **awk** will not use field separators any more, unless you assign a new value to FS.

awk allows you to assign values to the fields, and that assignment will prevail over the values assigned when **awk** reads the input record. Suppose the seventh field of a record contains the string scoobydoo, and you set it to another value:

```
$7 = "seven"
```

Then the seventh field will contain the string seven and the $0 will be recomputed to reflect the change.

As an example, and to review a number of the concepts above, let's study this **awk** program:

```
awk -F: '{
   OFS="."      # set the output field separator
   print $0     # print all input fields
   $1="first"   # re-assign the first field
   print $0     # print all input fields again
}' myfile
```

awk is called using the -F flag to set the field separator to a colon. We could have achieved the same result by assigning the value ":" to the FS variable. In our example, we assume the program has been entered at the command-line interface. Therefore we placed the program between quotes to prevent variable expansion by the shell. The main block of code is specified between curly braces, and no conditions or expressions are specified before that block, therefore it will be executed for any record.

In the main code block, we start by setting the OFS variable to a dot. This signifies that all output fields will be separated by a dot. Next, the input line is printed by referring to $0. Then, on the next line, we force the first field into a fixed value, and then we print out the (newly created) full record again. On the final line, the block is closed and the input file

name is specified. Assuming the file myfile contains these two lines:

```
a:b:c:d
1:2:3:4
```

The output would be:

```
a:b:c:c
first.b.c.d
1:2:3:4
first.2.3.4
```

It is possible to refer to non-existent fields or to assign values to non-existing fields, but it's beyond the scope of this introduction. See the manual pages for (GNU) awk for more details.

Branching, looping and other control statements

Within **awk** programs, you can use a number of control statements. Here, too, statements can be grouped together by using curly braces. A statement (or group of statements) can be prepended by a control statement. If the control statement evaluates to true, the corresponding block of code (or single statement) will be executed. For example the if statement has the form:

```
if (condition) statement
```

.. or, if blocks of code are used:

```
if (condition) { statements... }
```

A sample program that uses if and demonstrates the use else and the use of blocks of code follows:

```
if ($NF > 4) {
  print "This record has over 4 fields"
  print "It is a very lengthy record!"
  long++
} else {
  print "Never more than four - good :-)"
  short++
}
```

or, the simpler case, this fragment of code uses a single statement:

```
if ($NF < 3 ) print "less than 3 fields found"
```

Looping (iteration) can be done by using either the while, do while or for statement:

```
while (condition) statement
do statement while (condition)
for (expr1; expr2; expr3) statement
```

An example is given below. It prints out the numbers from 1 to 10, using all three methods. Additionally, the keyword exit is introduced. exit simply terminates the program. This program makes use of a simple trick to make it run without input data: it consists of just a BEGIN block (more about that later; patterns). Also note the use of the *increment operator* ++ (operators).

```
BEGIN {
  # first method:
  #
  number=1
  do {
     print number
     number = number + 1
  } while (number < 11)

  # second method:
  #
  number=1
  while ( number < 11 ) print number++

  # third method:
  #
  for (number=1; number<11; number = number + 1) print number

}
```

The exit statement terminates the program. It can be followed by an expression that should return a value from 0...255. This value is passed on to the calling program, for example, to enable it to see why a program terminated:

```
$ awk 'BEGIN { exit 56 }'; echo $?
56
```

Note

Only values between 0 and 255 are meaningful exit values, all other values will be converted into that range.

The break statement can be used to jump out of a do, while or for loop unconditionally. The loop is terminated, and program continues at the first statement after the loop. For example, here is another (somewhat clumsy) way to count to ten using **awk**:

```
BEGIN {
   z=0
   while (1==1) {
      if (++z == 10) break
      print z
   }
   print "10"
}
```

Under normal circumstances, all commands within a block of code executed under the control of either a while, do or for function will be executed for each iteration. The continue statement can be used to perform the next iteration, thereby skipping the code between the continue statement and the end of the loop. Yet another way to count to ten, for example, is:

```
BEGIN {
   z=0
   while (1==1) {
      print z;
      if (++z < 10) continue
      print "10"
      break
   }
}
```

Patterns

As stated before, **awk** programs are composed of blocks of code prepended by patterns. The blocks of code will be executed if the pattern applies.

Patterns can be in any of the following forms:

```
BEGIN
END
```

/regular expression/
relational expression
pattern && pattern
pattern || pattern
pattern ? pattern : pattern
(pattern)
! pattern
pattern1, pattern2

BEGIN and END. An **awk** program can have multiple BEGIN and END patterns. The actions in the BEGIN block will be executed before input is read, the actions in the END block after the input is terminated. Blocks of code that start with a BEGIN pattern are merged together into one virtual BEGIN block. The same applies to END blocks. Thus, a (somewhat unusable) interactive **awk** program like this:

```
BEGIN {
   print "1"
}
END {
   print "a"
}
BEGIN {
   print "2"
}
END {
   print "b"
}
{ exit }
```

.. assuming you would enter some input, would print:

```
1
2
.. some input you gave ..
a
b
```

Regular Expressions. For blocks prepended with a /regexp/ pattern, (see the section called "Regular Expressions") the code found in the corresponding block will be executed for each input record that matches the regular expression regexp specified. **awk** uses an extended set of regular expressions, similar to **egrep(1)** (remember that the *interval expressions* are either not supported at all or can only enabled on request).

Relational expressions. Another way to select which block to execute for which record is using *relational expressions*. Such an expression consists of **awk** commands (a.k.a. actions). Frequently, this is used to match on fields within a record:

```
$ awk '$1 == "01" { print }'
```

This would print all input records (lines) that have the string "01" as their first field. You can also check fields to match regular expressions, by using the "match" operator ~ (the tilde character), as is done in this one-liner:

```
$ awk '$1 ~ /[abZ]x/ { print $2 " " $3 }'
```

Given the following input:

```
Zx the rain
cx and snow
ax in Spain
gx and France
```

this would produce:

```
the rain
in Spain
```

Logical operators. Combinations of pattern expressions can be made by using logical operators && (and) || (or) and ! (not). As in C and **perl**, these work in a short-circuit fashion: if enough of the expression is resolved to determine the outcome (true or false) of the expression, the other patterns are not resolved. Thus, given this example:

```
( $1 > 0 ) && ( $2 < 42 )
```

if $1 was assigned the value 1, the first part of the expression would return "*true*". Because the resolution of the second expression, "($2 < 42)", is of no relevance to the outcome of the entire expression, it will never be executed. This example:

```
$ awk ' $1 == 3 || $2 ~ /[a-z]9/ { print }'
```

would print all records where the first field has the value 3, or, if the first field does *not* contain a string with the value 3, would check if the second field matches the regular expression [a-z]9.

Conditional operators. The conditional operator {pattern}?{pattern}:{pattern} (sometimes

called the *ternary operator*) works just like the same operator in C. If the first pattern is *true*, then the pattern used for testing is the second pattern, otherwise it is the third. Only one of the second and third patterns is evaluated.

```
$ awk '$1==3 ? $2=="b" : $2=="c" { print }'
```

This signifies, that, if the first field contains a string with value "3", the record will be printed if the second field contains the string "b". If the first field does *not* contain the value "3", the record will be printed if the second field contains the string "c". All other records are not printed.

Range operator. Finally, there is the form where a range of records can be specified. This is done by specifying the start of the range, a comma and the end of a range. In this example, selected records will be printed:

```
$ awk '$1=="start", $1=="stop" { print }'
```

This program will suppress lines in its input until a record (line) is found in the input where the first field is the string "start". That record, and all records following it, would be printed - until a record is found where the first field is the string "stop". Any records *following* that line will silently be discarded, until the next record is found where the first field is the word "start".

Operators

awk supports standard operators similar to those used in C. From the manual pages:

Table 13.13. awk operators

(...)	Grouping of statements, for example, to force order of execution
++ --	Increment and decrement, both prefix and postfix
^ **	Exponentiation. Both alternatives support the same functionality
+ - !	Unary plus, unary minus and logical negation
+ -	Addition, subtraction
* / %	Multiplication, division and modulus
(space)	the space is used for string concatenation
< <= > >= != ==	relation operators: lesser, lesser or equal, bigger, bigger or equal, not equal, equal
!~	match a regular expression
in	array membership
&& ||	logical and, logical or

?:	This has the form expr1 ? expr2 : expr3. If expr1 is true, expr2 is evaluated, else expr3 is evaluated.
= += -= *= /= %= ^= **=	assignments. The basic form is var = value. The operator form, for example, var operator= value is an alternate form to write var = var operator value, b *= 3 is the same as b = b * 3. Note that the form ^= is an alternate way of writing **=

Using regular expressions

The chapter on Regular Expressions (see the section called "Regular Expressions") gives a thorough overview of the regular expressions that can be used in either **awk**, **perl** or **sed**. This section, therefore, focuses on using them within **awk**, rather than on the *Regular Expressions* themselves.

A regular expression can be used as a pattern by enclosing it in slashes. Then the regular expression is tested against the entire text of each record. Normally, it only needs to match some part of the text in order to succeed. This, for example, prints the second field of each record that contains the characters "snow" anywhere in it:

```
$ awk '/snow/ { print $2 }' phonelist
```

Built-in variables

An exhaustive list of all built-in variables can be found in the manual pages for **awk**; only the most commonly used ones are explained here. The built-in variables FS and OFS have been demonstrated before. They can be used to force the Field Separator and Output Field Separator by setting them from within an **awk** code block. The FNR variable holds count of the current record number. A very basic one-liner to generate numbered program listings, for example, could look like this:

```
awk '{ printf "%05d %s\n",FNR, $0 }'
```

By default, records are separated by newline characters, that is, **awk** reads lines of text by default and treats each line as a record. This behaviour can be changed by setting the variable RS. RS contains either a single character or a regular expression. Text in the input that matches that character or regular expression is used as a separator. The environment can be read by using the contents of the internal array ENVIRON. To read the variable TERM for the environment, you would use ENVIRON["TERM"].

Functions

awk allows the definition of functions by the programmer and has a large number of built-in

functions.

Built-in functions

The built-in functions consist of mathematical functions, string functions, time functions and the system function.

The mathematical functions are: atan2(y, x), cos(expr), exp(expr), int(expr), log(expr), sin(expr) and sqrt(expr).

Additionally, there are two functions available to work with random numbers: srand([expr]), which seeds the random generator, and rand(), which returns a random number between 0 and 1.

A rich set of string functions is available; please consult the manual pages for a full description. A short list of the most commonly used functions follows: index(s,t) to return the index of the string t in the string s, length([s]) to obtain the length of the string specified (or the length of the input record if no string is specified), match(s,r) to return on which position in string s the expression r occurs. substr(s,i[,n]) returns the substring from string s, starting at position i, with optional length n.

One of the primary uses of **awk** programs is processing log files that contain time-stamp information. **awk** provides two functions for obtaining time stamps and formatting them: systime() returns the current time of day as the number of seconds since midnight UTC, January 1st, 1970, and strftime([format [, timestamp]]) to format time stamps according to the form specified in format. To learn more about the format string used with this function, the manual pages for strftime(3) should be consulted.

Self-written functions

User-defined functions were added later to **awk**, and therefore, the implementation is sometimes, somewhat, well, *awkward*. User-defined functions are constructed following this template:

```
function name(parameter list) { statements }
```

As an example, we'll list a function that prints out a part of a string (you may be familiar with it, since we've used it before in our examples):

```
function resume(line, part)
{
  len=length(line)
  if (len > ( 2 * part) )
  {
```

```
    retval = substr(line,1,part) " ... " substr(line,len-part)

  } else {

    retval = line
  }
  return retval
}
```

This function is named "resume" (*reh-suh-may*). It prints a short version of its input line, consisting of an equal number of characters from the begin and end of the line, concatenated by an ellipsis. It has two parameters:

- line, the line to determine the resume from, and
- part, which is the exact number of characters counted from the beginning and the end of the line that should be displayed.

Within the function, the parameter will be available as (local) variables line and part. This function returns its result, which in this case will be a string.

Arrays are passed to functions *by reference*, therefore, changing the value of one of the members of an array will be visible in the other parts of the program, too. Regular variables are passed by value, therefore when a function alters its value, that is not automatically visible to other parts of the program. For example:

```
function fna(a) { a[0]=3 }
function fnb(b) { b=4 }
BEGIN {
      a[0]=4
      fna(a)
      print a[0]

      b=8
      fnb(b)
      print b
}
```

This would print

```
3
8
```

Note that whilst defining a user-defined function, the name should be followed immediately

by the left parenthesis, without any white space.

Functions are executed when called from within expressions in either actions (see the example above) or patterns. An example of usage of a function in a pattern follows:

```
function has_length_of (string, len)
{
   return (length(string) == len)
}

has_length_of($0,4) { print "four" }
```

This will print out the string "four" for each line that contains exactly 4 characters.

The provision for local variables is rather clumsy: they are declared as extra parameters in the parameter list. The convention is to separate local variables from real parameters by extra spaces in the parameter list, e.g.:

```
function  f(p, q,     a, b)   # a & b are local
{
    ... code ...
}

{ f(1, 2) }
```

Functions may call each other and may be recursive. To return a value from a function, you should use return expr. The return value will be undefined if no value is provided.

---

rsync

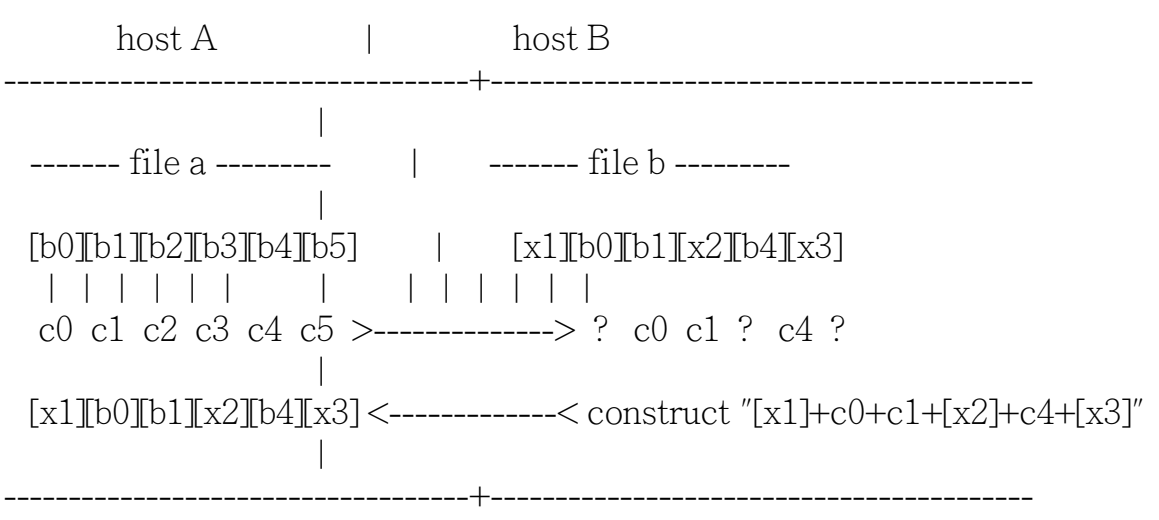
rsync is an open source utility that provides fast incremental file transfer. **rsync** is a replacement for **rcp**, but has many more features. **rsync** is mostly used to synchronize files between systems. It does this very efficiently, for it uses a special algorithm that just sends the differences between the files over the link.

The rsync algorithm

To determine the differences between two files, without the need to have both files available on one host, one of the hosts divides its version of the file to **rsync** in fixed-size blocks. For each block, a checksum is calculated[23]. The checksums are sent to the other side. The other host scans its version of the file for occurrences of blocks with matching checksums (even if these blocks are not located at the same boundaries). With more or less similar files, chances are that many matching blocks will be located.

After completion of the analysis, the host sends its peer information on how to construct a copy of the file in terms of blocks with literal data or the identifier for a common matching block it has found.

Figure 13.1. rsync protocol simplified



In the figure above, at the left, host A is represented. It has a file `a`, an older version of file `b`. To synchronize the two, host A will split up its file into blocks (in our example: [b0]..[b5]) and calculate checksums over these blocks (c0..c5). Then, it will send these to the other host (host B). Host B will figure out if any blocks can be found in "old" file `b` that have the same checksum. Since both sides probably have a number of blocks in common, these parts of the file do not need to be sent over the line. Host B will tell host A how to construct file `b` by either sending over blocks of literal data (for blocks that could not be matched anywhere) or the identifier of a block that both

sides have in common.

Configuring the rsync daemon

rsync can also talk to "*rsync servers*", which can provide anonymous or authenticated **rsync**.

Synchronization is done between two machines. One of these needs to run a server-program, the other can use **rsync** in client-mode. The **rsync** daemon can be run in standalone mode:

```
# rsync --daemon
```

Or you can make **inetd** start it, in which case you will need to add a line to its configuration file (/etc/inetd.conf):

```
rsync stream tcp nowait root /usr/bin/rsync rsyncd --daemon
```

rsync is capable of using **ssh** or **rsh** for transport. Using **ssh** has the advantage that all traffic will be properly encrypted and authentication can be done transparently, using RSA/DSA public keys (see the section called " Secure shell (OpenSSH) (2.212.4) "). **rsync** can be used stand-alone too, in which case the data is sent over port 873. Ensure that this port is listed in the file /etc/services as "rsync".

The daemon uses the configuration file /etc/rsyncd.conf. An example follows:

```
lock file = /var/run/rsync.lock
pid file = /var/run/rsyncd.pid
log file = /var/log/rsyncd.log
motd file = /etc/rsyncd.motd

[openarea]
  path = /fs0/rsync/public
  comment = Public Area
  uid = nobody
  gid = nobody
  read only = no
  list = yes
  hosts allow = jones smith
  auth users = peter wendy
  secrets file = /etc/rsyncd.scrt
  use chroot = yes
```

The first four lines globally specify the location of various files the **rsyncd** daemon uses. These are the *lockfile* (lock file), the file that will contain the process ID for the process (pid file), the file in which to log messages (log file) and the file in which a message of the day file (motd file) can be found. The latter is displayed if a client connects to the **rsync** server.

An *area* is nothing more that a grouping of options for access to certain areas on your server.

Sometimes areas are referred to as *modules* or (somewhat confusingly) *paths*. Options are specified within an area and only valid within that area. The configuration file can have many area-definitions, ours just has one.

An area is denoted by its name within square brackets: such as [openarea]. The path option defines the path where the files are **rsync**'ed to and from; the comment option specifies a descriptive explanation for listings; the uid and gid specify which uid/gid the server will use to access the area. The authorized users are listed after the auth users option. These are the names of users allowed to access this area. The auth users do not have to be system users, however, their names should occur in the *secrets file*. This is a file containing plaintext key/value pairs of usernames and passwords. The location of that secrets file can be specified using the secrets file option. The option hosts allow denies access to all hosts other than those whose IP address or hostnames are listed. Its counterpart hosts deny does the opposite: it allows everybody, except the hosts whose IP address or hostnames are listed. The other options specify that the file-system path is read/write (read only) and the **rsync** path shows up in **rsync** listings (list). For security reasons, the rsync daemon can chroot to the path specified, which is switched on by the "use chroot" option. Many more options can be found in the man page for **rsyncd.conf**.

Using the rsync client

The **rsync** client can be used with either **rsh** or **ssh**. **ssh** is recommended because it encrypts communication (for privacy and security reasons).

The client has many options. The manual page lists them all and contains plenty of good examples. Just to give you an impression, here is an example of typical use:

```
$ rsync -avzrpog --rsh=/usr/bin/ssh --compress henk@rsync.opensource.nl::lpic2book /fs0/docbook/lpic2
```

As you can see, there are two command line option notation methods, the old-style single minus prefixed notation and the posix-compliant double-minus prefixed notation. This will retrieve the files in the area lpic2book on server rsync.opensource.nl and put them in /fs0/docbook/lpic2 on the local server, using the username henk to log in. The options -avzrog are used to get the data in an archive, to set verbose mode, fetch the files recursively, and preserve the permissions, owner and group used on the original files. Another example:

```
$ rsync rsync.opensource.nl::
```

.. will list out all the modules running on the server.

rsync is capable of using include and exclude patterns, which have a specific syntax (somewhat like shell wildcards, but with exclusions and extensions).

---

[23] Actually, *two* checksums: one strong 128 bit MD4 checksum, and a special so-called "rolling checksum", which is not accurate per se, but has special properties that enable easy detection of a matching block even when that block is not on a block boundary. If a matching "rolling checksum"

is found, a MD4 checksum is calculated over it and compared to the original MD4 checksum to ensure that the block found literally matches its counterpart.

Prev	Up	Next
Using awk	Home	crontab

crontab

Unix systems start up the **cron** daemon in multi-user mode. Hence, so does Linux. The **cron** daemon will execute programs at predestined times. The daemon wakes up every minute and runs the commands as necessary.

Files in which the time and interval to run commands are specified, as well as the commands themselves, are called **crontab** files. All **cron** implementations support user **crontabs**: each system-user has his/her own **cron** command file. These individual configuration files are stored in a standard directory: /var/spool/cron/crontabs or /var/cron/tabs/. They bare the name of the appropriate user-account from the /etc/passwd file.

By default, either only the superuser or every user will have the right to create and maintain a crontab file (which of the two depends on the local configuration). For a more fine-grained authorization scheme, *either* the file /etc/cron.allow or the file /etc/cron.deny can be created by the superuser. Such a file contains lines with usernames. If the cron.allow file is found, only users that are listed in that file will have the right to maintain entries, if the cron.deny file is found, all users *except* the users listed there can use cron.

The information in user files is often stored in memory by the daemon - therefore, when you change the user files manually, by editing them, you will need to restart the **cron** daemon to activate the changes. However, under normal circumstances you should not edit the user files by hand, but use the command

```
$ crontab -e
```

to edit your personal entry. This will start up your editor, enable you to edit your entry and will activate the changes if your entry could be validated. The file /etc/crontab and the files in the /etc/cron.d directory are monitored by the daemon for changes.

Alternately, you can use a file in your HOME directory (or another location where you have write-access) and edit this file. To active it, use

```
crontab <the-file-name>
```

To deactivate it, use:

```
crontab -r
```

This has the advantage over the **crontab -e** method that your local copy of the **cron**tab file remains available for future use.

Modern implementations of **cron** often look in more places for configuration files. First of all, there

is the file /etc/crontab. It can be edited by root and allows specification of a user ID to use when executing the command. Additionally, **cron** will concatenate any files found in /etc/cron.d and will treat them as if they were part of (extensions to) /etc/crontab.

Most distributions also offer provisions to enable easy execution of commands that need to be run on an hourly, daily, weekly or monthly basis. This is done by creating directories named /etc/cron.hourly, /etc/cron.daily and so on, in which directories scripts can be placed. These are run sequentially, both to preserve system resources and to be able to support dependencies between scripts ('this has to run first, before that can be run'). Typically, the order of execution is determined by the name of the script; therefore, their names are often prepended with equal-length short-strings to force the correct execution order, for example: aaa_logoutusers or aab_cleanfs. Often, the execution of these scripts is initiated by a script that in turn is started by a **crontab** entry in the /etc/crontab file.

When executing commands, a default environment setting is provided, containing at least the HOME, PATH and TERM variables. The PATH often is set to very minimalist values. Any output is mailed to the owner of the crontab or to the user named in the MAILTO environment variable, if it exists.

format of the crontab file

You can define or override environment variables, by defining them in the crontab file, one variable per line, in the format:

```
VARIABLE=value
```

Lines whose first non-whitespace character is a hash (#) are comment-lines. Empty lines are discarded. The format for lines in a crontab entry is:

```
minute hour day-of-month month day-of-week (who-and-)what-to-do
```

The minute field can have values from 0-59, the hour field 0-24, the day-of-month field 1-31, and the day-of-week field 0-7; both 0 and 7 signify Sunday, 1 is Monday, and so on. The (who-and-)what-to-do field contains either just the command and its parameters (in case this is an entry in an individual crontab entry), or the name of the user ID used to run the command, followed by the command and its parameters. A star signifies "don't care" or "always match". If you want to run the local command **something** every minute[24], with parameter -else and redirection of the output to the null-device, you might specify:

```
* * * * * /usr/local/bin/something -else >/dev/null 2>&1
```

Additionally, you are allowed to specify lists of values and ranges to match. A range is a (inclusive) specification of initial and final values, separated by a hyphen. So, for example, 4-8 specifies that all numbers in the set 4, 5, 6, 7 and 8 will match. A comma-separated list of values and/or ranges is allowed, too. Therefore, 1-3,6 would translate to *matching either 1, 2, 3, or 6*, and 1-3,5-8,10 would translate to *matching either 1, 2, 3, 5, 6, 7*, 8 or 10'. Some examples:

```
0 5 * * * find $HOME -type f '(' -name core -o -name dead.letter ')' -atime +7 -mtime +7 -exec rm -f '{}' ';'
```

This is an entry in a user crontab, that scans the HOME directory and below for files named dead.letter and core, that have not been accessed for at least a week and removes them.

```
8,16,24 8-13 * * 1-5 uucp /usr/local/bin/poll
```

This is an example entry that could occur in /etc/crontab or in one of the files in /etc/cron.d. It makes **cron** start up the command **/usr/local/bin/poll** on Monday through Friday (the fifth field, 1-5), at 8, 16 and 24 minutes past the hour, for all hours between 8 and 13 hour inclusive. The first time the command will be executed during a week is on Monday, at 8h08, and the last time on Friday, on 13h24. This command will be run with the UID of the user uucp.

You can use names instead of numbers for weekdays and months, but this feature is not fully implemented; therefore specifying ranges using these names are *NOT* allowed. Additionally, you need to specify the first three letters of the days, more, no less. Following a range with a slash results in matching with the members of the range, but not with the default step value "1". So, 0-10/3 signifies the values 0, 3, 6 and 9.l To say "every 4 hours", you could use the construct "*/4".

The at command

Closely associated with **cron** (and often implemented by it) is the command **at**. Using this command, you can execute a given command or set of commands at a later time. However, unlike **cron**, the commands executed with **at** are only executed once. The time and date for execution can be specified on the command line; the command(s) needs to be specified on standard input:

```
$ at {given-time}
command-1
command-2
^D
$ _
```

The time can be constructed using many forms, the simplest is by specifying the hour and minute (time) you want the commands to run (hh:mm). The time will always be assumed to be in the future. A number of keywords may are also available: midnight, noon or even teatime (16.00 hours). You can use either 24 hour notation, or append "PM" or "AM" to the time. To specify a date you can use the form month day or month day year, (e.g., Jul 17 2038) or the form "MMDDYY", "MM/DD/YY" or "DD.MM.YY". To specify both time and date, *first* specify the time, and then the date:

```
$ at 20:00 Jul 17 2038
```

Relative time is acceptable. "In 20 minutes" can be specified as follows:

```
$ at now + 20 minutes
```

Instead of minutes, you can also specify hours, days or weeks. To tell **at** to run a job tomorrow you just say so:

```
$ at now + 20 minutes tomorrow
```

The superuser may use **at**. Other users' permission to use it is determined by the files /etc/at.allow and /etc/at.deny, which work much the same as the previously described files /etc/cron.allow and /etc/cron.deny. If the allow-file exists, only usernames mentioned in it may use **at**. If the allow file does not exist, /etc/at.deny is checked and every username *not* mentioned there is allowed to use **at**. An empty /etc/at.deny means that no one is restricted.

---

[24] assuming your **cron** wakes up every minute.

Monitoring your system

Resources: Prasad01

As we have seen, Unix systems contain many tools: tools to start up programs automatically, synchronize files between systems and help manipulate data (**sed**, **awk** and **perl**). You may have noticed that there is a huge overlap in functionality between these three. Why do we have **sed**, if **awk** has most of its functionality too? Why do we need **awk** when we can use **perl** instead? Well, there are many reasons. First of all, you need to realize that it just grew this way. **sed** was there before **awk** and **perl** came after **awk**. But, as there are still many **sed** and **awk** scripts around that offer very useful functionality, both tools are still maintained and heavily used. There is also the question of resources: the overhead needed for a simple **sed** script is much less than that of a full-blown environment like **perl**'s. Also, a number of people just feel more comfortable using one set of tools, while others like another set. And, as a sysadmin or poweruser, it is very convenient to have alternate methods available if parts of the system are damaged. Say, for example, that **ls** does not work (because you are in a **chroot**ed environment or somebody inadvertently deleted it), in which case you can use the shells' expansion mechanism instead:

```
echo *
```

In practice, it all boils down to your personal preference and the environment you have available. However, it really helps to be able to work with *regular expressions* (see the section called "Regular Expressions"), the salt of all Unix systems. Study them well.

In the next sections, we will provide various examples of scripts that can be used to monitor your system, using various toolsets. We will give examples of various ways to parse a log-file, combine log-files and generate alerts by mail and pager. We will write a script that monitors files for changes and another that warns administrators when specified users log in or out.

Warning

The scripts we provide are meant to be examples. They should not be used in a production environment "as is", since they are written to clarify an approach; they are intended to help you understand how to write solutions for system administration problems using the basic Unix components. The scripts lack proper attention security and robustness, the scripts, for example, do not check whether a statement has completed successfully; signals that may disrupt the scripts are not caught, sometimes modularity is offered to support didactical

clarity etc.

parsing a log-file

As a system administrator you often will study the contents of logfiles. Logfiles are your main source of information when something is not (quite) working. Alas, often these logfiles are riddled with information you do not need (yet). Therefore, it is useful to be able to filter the data. For example, the **syslogd** daemon logs so-called *marks* - lines containing the string "-- MARK --". It pumps out one such line every 20 minutes[25]. This is useful to check whether or not the daemon is running successfully, but adds a huge number of lines to the logfile. You can remove these lines in many ways: Unix allows many roads to a given goal. A simple **sed** script could do the trick:

```
sed '/-- MARK --$/d' /var/log/messages
```

or you could use another essential Unix tool:

```
grep -v -- "-- MARK --" /var/log/messages
```

Note the double dash that tells **grep** where its options end. This is necessary to prevent **grep** from parsing the searchstring as if it were an option. Yet another way to obtain the same goal:

```
awk '!/-- MARK --$/ { print }' /var/log/messages
```

And finally, using **perl**:

```
perl -ne '/-- MARK --$/ || print;' /var/log/messages
```

Or, in the true tradition of **perl** that even within **perl** there is more than one way to do things:

```
perl -ne 'print unless /-- MARK --$/;' /var/log/messages
```

Another example: suppose you are interested in those lines from the messages file that were written during lunch breaks (noon to 14.00). You could filter out these lines using sed:

```
sed -n '/^[A-Z][a-z]\{2\} [ 1-3][0-9] 1[23]:/p' /var/log/messages
```

The -n option tells **sed** not to print any lines, unless specifically told so using the 'p' command. The regular expression looks like stiff swearing on first sight. It isn't. Ample analysis reveals that it instructs **sed** to print all lines that begin (^) with an uppercase letter ([A-Z]), followed by 2 lowercase letters ([a-z]\{2\}), a space, either a space or one of the digits 1-3 ([1-3]), yet another digit ([0-9]), another space, the digit 1 and either the digit 2 or the digit 3 ([23]), followed by a colon. Phew.

Or you could use this **awk** one-liner:

```
awk '/^[A-Z][a-z][a-z] [ 1-3][0-9] 1[23]:/' /var/log/messages
```

Note that this time we did not use the interval expression ({2}), but simply duplicated the expression. By default **gawk** (which we are using throughout this book) does not support interval expressions. However, by specifying the --posix flag it can be done:

```
awk --posix '/^[A-Z][a-z]{2} [ 1-3][0-9] 1[23]:/' /var/log/messages
```

Alternately, you could use a simple **awk** script:

```
awk '{
      split($3,piece, ":")
      if ( piece[1] ~ /1[23]/ ) { print }

}' /var/log/messages
```

which renders the same result, using a different method. It simply looks for the third (white-space delimited) field, which is the time-specification, splits that into 3 subfields (piece), using the colon as delimiter (split) and instructs **awk** to print all lines that have a match (~) with the third field from the timefield.

Finally, we can use **perl** to obtain the same results:

```
perl -ne '/^[A-Z][a-z]{2} [ 1-3][0-9] 1[23]:/ && print;' /var/log/messages
```

As an exercise, you could try some other approaches yourself. If you want to be reasonably sure that your *do-it-yourself* approach renders the same result as the methods tested above, make a copy of (a part of) the messages file, run one of the commands we gave as an example on it and pipe the result through **sum**:

```
$ cp /var/log/messages ./myfile
$ sed -n '/^[A-Z][a-z]\{2\} [ 1-3][0-9] 1[23]:/p' ./myfile |sum
48830   386 [26]
```

Your home-brewn script should render the exact same numbers you get from one of the example scripts when its results are piped through **sum**.

combine log-files

Often, you'll need to combine information from several files into one report. Again, this can be done in many ways. There is no such thing as *the right way* - it all depends on your skills, preferences and the available toolsets. You are free to choose whichever method you want. However, to get you started, some examples follow.

The command **groups** will list the groups you are in. It accepts parameters, which should be usernames, and will display the groups for these users. By the way: on most systems **groups** itself is a shell script. An example of typical use and output for this command follows:

```
$ groups nobody root uucp
nobody : nogroup
root : root bin uucp shadow dialout audio nogroup
uucp : uucp
$ _
```

The information displayed here is based on two publicly readable files: /etc/passwd and /etc/group. The **groups** command internally uses **id** to access the information in these files, but nothing stops us from accessing it directly. The following sample shell script consists mainly of an invocation of **awk**. It will display a list of all users and the groups they are in, and, as an extra, prints the full-name of the user.

```
  /--------------------------------------------------------------------
01 | #!/bin/sh
02 | #
03 |
04 | # This script displays the groups system users
05 | # are in, on a per user basis.
06 |
07 | awk -F: 'BEGIN {
08 |
09 |       # Read the comment field / user name field from the
10 |       # passwd file into the array 'fullname', indexed by
11 |       # the username. If the comment field was empty, use
12 |       # the username itself as 'comment'.
13 |       #
14 |       while (getline < "/etc/passwd") {
15 |             if ($5=="") {
```

```
16 |                 fullname[$1] = $1
17 |             } else {
18 |                 fullname[$1] = $5
19 |             }
20 |         }
21 | }
22 |
23 | {
24 |         # Build up an array, indexed by user, that contains a
25 |         # space concatenated list of groups for that user
26 |
27 |         number_of_users=split($NF,users,",")
28 |
29 |         for(count=0; count < number_of_users; count++) {
30 |             this_user=users[count+1]
31 |             logname[this_user] = logname[this_user] " " $1
32 |         }
33 | }
34 |
35 | END {
36 |         # List the array
37 |         #
38 |         for (val in logname) {
39 |             printf "%-10s %-20s %s\n",
40 |                 substr(val,1,10),
41 |                 substr(fullname[val],1,20),
42 |                 logname[val]
43 |         }
44 | }' /etc/group
  \-----------------------------------------------------------------------
```

An example of how the output might look:

```
at        at                  at
bin       bin                 bin
db2as     DB2 Administration  db2iadm1
named     Nameserver Daemon   named
oracle    Oracle User         dba
db2inst1  DB2 Instance main us  db2asgrp

... output truncated ...
```

On line 01 we state that the interpreter for this script is /bin/sh. That may seem surprising at

first, since the script actually consists of **awk** statements. However, there are two exceptions: the command-line parameter (-F:), (line 07) and the filename /etc/group (line 44). It is possible to rewrite this program as a pure **awk** script - we leave it up to you as an exercise. The **awk** construction consists of three groups:

- the BEGIN group, which builds up an array containing the full-names of all system users listed in /etc/passwd,
- the body, which reads the lines from /etc/group and builds an array that contains the groups per user
- the END group, which finally combines the two and prints them.

the BEGIN block. Line 14 shows a method to sequentially read a file *within* an **awk** program block: the getline function. getline fetches an input-line (*record*) from a file and, if used in this context, will fill the variables (e.g., $0..$NF) accordingly. By putting getline in the decision part of a while loop, the entire file will be read, one record (line) per iteration, until getline returns 0 (which signals "end of file") and the while loop is terminated. In our case, the file /etc/passwd is opened and read, line by line. We set the field-delimiter to a colon (-F:, line 07), which is indeed the field-separator for fields in the /etc/passwd file. Within the loop we now can access the username ($1) and full name of (or a comment about) that user ($5). The construct used on line 16 (and 18) creates an array, indexed by *username*.

the main block. In the main block, **awk** will loop through the /etc/group file (which was specified on the command line as input file, line 44). The variable $NF always contains the contents of the *last* field, which, in this case, is a comma-delimited list of users. Line 27 splits up that field and puts the usernames found in the last field into an array users. The for loop on lines 29-32 adds the group to the proper elements of array logname, which is indexed by username.

the END block. The END block will be executed when the input-file is exhausted. The logname array was previously indexed by username: each array-element contains a space-delimited list of group-names (e.g., the array-element logname[root] could contain the string "root bin shadow nogroup"). The "for (val in logname)" loop (line 38) will walk through all elements of the array logname, whilst putting the name of the element in the variable val. Line 39 defines the output format to use: first a left-aligned string-field with a length of 10 characters, if necessary padded to the left with spaces (%-10s); next, another left-aligned string-field with a length of 20 characters, also padded to the left with spaces (%-10s), and finally a string of variable length. To prevent field overflow, the substr function is used to trim fields to their maximum allowed length.

Of course, you can do the very same thing using a **perl** script. You can start *from scratch*, but since we already have a fully functional **awk** program that does the trick, we can also use a utility to convert that script to a **perl** program. On most Unix systems it's named **a2p**. It takes the name of the **awk** script as its argument and prints the resulting output to *standard output*:

a2p userlist.awk > userlist.pl

The results are often pretty good, but in our case some fine-tuning was needed, both to the originating and resulting scripts, since the **awk** program in our example was embedded in a shell-script. The example below lists our result:

```
  /--------------------------------------------------------------------
01 | #!/usr/bin/perl
02 |
03 | open(_ETC_PASSWD, '/etc/passwd') || die 'Cannot open file "/etc/passwd".';
04 | open(_ETC_GROUP,  '/etc/group') || die 'Cannot open file "/etc/group".';
05 |
06 | $[ = 1;                 # set array base to 1
07 |
08 | $FS = ':';
09 |
10 | # Read the comment field / user name field from the
11 | # passwd file into the array 'fullname', indexed by
12 | # the username. If the comment field was empty, use
13 | # the username itself as 'comment'.
14 | #
15 | while (($_ = &Getline2('_ETC_PASSWD'),$getline_ok)) {
16 |     if ($Fld[5] eq '') {
17 |       $fullname{$Fld[1]} = $Fld[1];
18 |     }
19 |     else {
20 |       $fullname{$Fld[1]} = $Fld[5];
21 |     }
22 | }
23 |
24 | while (($_ = &Getline2('_ETC_GROUP'),$getline_ok)) {
25 |     chomp; # strip record separator
26 |     @Fld = split(/[:\n]/, $_, 9999);
27 |
28 |     # Build up an array, indexed by user, that contains a
29 |     # space concatenated list of groups for that user
30 |
31 |     $number_of_users = (@users = split(/,/, $Fld[$#Fld], 9999));
32 |
33 |     for ($count = 0; $count < $number_of_users; $count++) {
34 |       $this_user = $users[$count + 1];
35 |       $logname{$this_user} = $logname{$this_user} . ' ' . $Fld[1];
36 |     }
```

```
37 | }
38 |
39 | # List the array
40 | #
41 | foreach $val (keys %logname) {
42 |     printf "%-10s %-20s %s\n",
43 |     substr($val, 1, 10),
44 |     substr($fullname{$val}, 1, 20),
45 |     $logname{$val};
46 | }
47 |
48 | sub Getline2 {
49 |     ($fh) = @_;
50 |     if ($getline_ok = (($_ = <$fh>) ne '')) {
51 |       chomp;  # strip record separator
52 |       @Fld = split(/[:\n]/, $_, 9999);
53 |     }
54 |     $_;
55 | }
  \-----------------------------------------------------------------------
```

As you might expect, this code works in a manner similar to the previously used **awk** example. The most striking difference is the addition of the Getline2 function. This function reads a line from the file specified by the file handle (which was passed as argument) and splits the records into fields, which are put in the array Fld. The chomp function (line 51) removes the record separator from the input-line, since **perl** does not do that automatically (**awk** does). Also, the **perl** program was rewritten to remove its dependency on a shell, therefore, the program opens *both* input files within (lines 03 and 04). Line 06 sets the array base to 1, which is the default for **awk** arrays, as opposed to **perl** arrays, which are base 0. The loop in line 15-22 reads the information from the passwd file and puts the full user names in the array fullname again, the loop on lines 24-37 reads in the group-file and builds the array that contains the lists of groups for each user. Finally, the code in lines 41-46 is almost identical to that of the END block in the **awk** example (lines 38-44) and works likewise.

The automatic generation of code does not necessarily deliver very clean code. In our example, some code was included that is not necessary to make this program work properly. As an exercise, try to find those lines.

generate alerts by mail and pager

In most cases, it suffices to construct reports and mail them to the system administrator. However, when an event occurs which requires immediate attention another mechanism should be used. In these cases, you could use SMS-messaging or page the sysadmin. You also could send a message to a terminal or have a pop-up box appear somewhere. Again,

there are many ways to achieve your goal. SMS and (alphanumeric) pagers are used most frequently.

Sending mail to somebody on a Unix system from within a process is very simple provided that you have set up e-mail correctly, of course. For example, to mail from within a shell program, use this command:

```
echo "Hi there!" | mail jones -s "Hi there!"
```

Or, to send longer texts, you could use a "*here document*" (described in the section called "Here documents"):

```
mail jones -s "Hi there!" <<EOF

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

EOF
```

To mail something from within Perl, a similar concept can be used:

```
open(MAIL , "|mail jones -s \"Hi There!\"") || die "mail failed";
print MAIL "Hi, Paula.";
close(MAIL);
```

To print more than one line, you can use may different techniques, for example, you could use an array and print it like this:

```
@text[0]="Hello, Paula\n";
@text[1]="\n";
@text[2]="Greetings!\n";

open(MAIL , "|mail jones -s \"Hi There!\"") || die "mail failed";
print MAIL @text;
close(MAIL);
```

Note that you need to insert newline characters at the end of the array elements if you want to start a new line. Alternatively, you could use a "*here document*"), as we did in the shell:

```
open(MAIL, "|mail root -s \"Hi There!\"") || die "mail failed";
print MAIL <<EOF;

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

EOF
close(MAIL);
```

Pager-service providers provide PSTN gateways: you dial in to these systems using a modem, over public phone-lines. Some special software needs to be written or downloaded to enable you to page someone. Often some kind of terminal based protocol is used. Other, systems use a protocol called TAP (Telocator Alpha-entry Protocol), formerly known as PET (Personal Entry Terminal) or IXO (Motorola invented it, including its name). Sometimes (in Europe) the UCP protocol (*Universal Communication Protocol*) is used. There are a number of software programs for Unix systems to enable them to use *PSTN* gateways, the most frequently used ones are **Hylafax**, **beepage**, **qpage**, **tpage** and **sendpage**. The latter is written in Perl and released under the GPL.

Pager-services are often replaced by SMS services nowadays, especially in Europe, where the GSM network is available almost everywhere. SMS offers the advantage of guaranteed delivery to the recipient. Additionally, most system administrators already have mobile phones. If SMS is used, they do not need to carry around an additional pager.

To enable computers to send SMS messages, special add-on GSM devices are available. They either accept serial input or are placed in a PCMCIA slot. They are capable of sending SMS messages directly. Some software has been written for them: you may want to look at http://smslink.sourceforge.org. By adding a computer that addresses such a device to your network, you can set up your own gateway(s) for your own network.

Apart from installing your own software/gateway, you can use external gateways, some of which are available on the Internet, others are provided by your Telecom operator:

- Web gateways. You just fill in the proper fields of the form and the message will be sent;
- SNPP (Simple Network Paging Protocol) gateways. These are capable of understanding

SNPP (see also RFC-1645). One way to implement such a gateway on your own network is by using the **sendpage** software;
- mail gateways. The method of usage varies from gateway to gateway. Many times, the message can be put in the body of the mail and messages are addressed using the pager number as the address for the recipient (e.g., <pagernumber>@<sms-gateway-domain>);
- most (Telecom/pager service) providers provide PSTN to SMS gateways. You dial in to these systems using a modem. Some special software needs to be written or downloaded to enable you to communicate with the gateway. Often either UCP or TAP protocols are used.

user login alert

The objectives state that you should be able to write a script that notifies (the) administrator (s) when specified users log in or out. There are, as always, many ways to accomplish this.

One solution would be to give a user a special login "shell", that issues a warning before starting up the "real" shell. This could be a simple shell-script (don't forget to **chmod +x** it) and named something like /bin/prebash. An example of such a script:

```
#!/bin/bash
#
DATE=‘/bin/date‘
#
mail root -s "USER $USER LOGGED IN" <<EOF

Attention!

The user $USER has logged in at $DATE.
EOF
exec /bin/bash --login
```

The corresponding line in the /etc/passwd file for this user could look like this:

```
user:x:501:100::/home/user:/bin/prebash
```

When the user logs in, the initiating script will be run. It will send the mail and then overwrite the current process with the shell. The user will never see that the warning has been issued, unless he or she checks out his/her entry in the /etc/passwd file. Alternately, you could choose to leave the "initiating" shell in place (i.e., not to issue the **exec** command) which lets you use the initiating shell to report to you when a user logs out:

```
#!/bin/bash
#
```

```
DATE=‘/bin/date‘

/usr/bin/mail root -s "USER $USER LOGGED IN" <<EOF

Attention!
The user $USER has logged in at $DATE.
EOF

/bin/bash --login

DATE=‘/bin/date‘
nohup /usr/bin/mail root -s "USER $USER LOGGED OUT" 2>/dev/null 1>&2 <<EOF2

Attention!

The user $USER has logged out at $DATE.
EOF2
```

This script consists of three parts: a part that is executed *before* the actual shell is executed, the execution of the shell and a part that will be executed *after* the shell has terminated. Note that the last mail command is protected from hangup signals by a **nohup** statement. If you leave out that statement, the mail process will receive a terminating signal before it would be able to send the mail.

Now, root will receive mail every time the user logs in or out. Of course, you can issue any command you want instead of sending **mail**. Note that the user could easily detect this guardian script is running by issuing a **ps** command. He could even kill the guardian script, which would also terminate his script, but the logout notification would never arrive.

Another method would be to invoke **logger**, which logs a warning to the syslog facility (e.g., for a script that just warns you when somebody has logged in):

```
#!/bin/bash
#
/usr/bin/logger "logged in"
exec /bin/bash --login
```

logger, by default, will log a line to the file that has been connected to the "user.notice" **syslog** facility (see: syslog.conf(5) and logger(1)). Note, that **logger** automatically prepends the line in the logfile with the date and the username, hence you do not need to specify it yourself.

We have described two ways to record a user logging in or out, using the Bourne shell to glue standard Unix/Linux commands together. There are many more possibilities: to use a

script that acts as a daemon and that regularly checks the process-table of any processes run on behalf of the user or to add a line to /etc/profile that alerts the sysadmin. All of these methods have various pros and cons. As usual, there are many ways to achieve what you want.

To demonstrate the use of **awk** to parse information, we will tackle this problem another way. On Linux systems, the **last(1)** command shows a listing of the logging behavior of users. The command obtains its information from the wtmp file (often found in /var/log). The wtmp file is used by a number of programs to records logins and logouts; **login**, **init** and some versions of **getty**. An example of its output follows:

```
piet    tty1                Tue Nov 27 18:04   still logged in
piet    tty1                Tue Nov 27 18:03 - 18:03  (00:00)
henk    :0         console      Thu Nov 15 16:38   still logged in
henk    :0         console      Wed Oct 24 17:02 - 18:33 (8+02:31)
reboot  system boot  2.2.18        Wed Oct 24 17:01        (8+02:33)

... lines truncated ...

wtmp begins Fri Sep  7 17:48:50 2001
```

The output displays the user, the (pseudo)terminal the user used to log in, the address of the host from which the user made the connection (if available) and finally a set of fields that specify how long the user has been logged in.

By running **last** every - let's say - five minutes and checking if anything has changed since the last time we checked, we can generate a report that tells us who has logged in/out since the last time we checked. An outline of the flow:

```
(begin)
  |
  +- store current situation in a file
  |
  <is there a history file?>
  |
 / \
n   y
|   |
|   +- make a diff between history file and new file
|   |
|   <were there changes since last time?>
|   |
|  / \
| n   y
```

```
    | |   |
    | |   +- mail report
    | |   |
    |  \ /
    |   |
     \ /
      |
      +- put current situation into history file
      |
    (end)
```

and a possible implementation, marked with line-numbers for later reference:

```
  /-------------------------------------------------------------------
01 | #!/bin/sh
02 |
03 | # The base directory to put our logs in
04 | #
05 | BASEDIR=/var/log/checklogin
06 |
07 | # The name of the history- and current file, in which the
08 | # login information is stored
09 | #
10 | HISTORY=${BASEDIR}/history
11 | CURRENT=${BASEDIR}/current
12 |
13 | # A simple generic failure function, aborts after displaying
14 | # its arguments.
15 | #
16 | fail()
17 | {
18 |    echo "Failed: $*"
19 |    exit 1
20 | }
21 |
22 | # This function cleans up the output of 'last', by
23 | # eliminating empty lines, reboot lines and wtmp
24 | # lines.
25 | #
26 | clean_last()
27 | {
28 |    /usr/bin/last | sed '{
29 |      /^reboot /d
```

```
30 |     /^$/d
31 |     /^wtmp begins /d
32 |   }'
33 |
34 | }
35 |
36 | MYGROUP=`id -gn`
37 | MYIDENT=`id -un`
38 |
39 | # Check our environment:
40 | #
41 | [ -d ${BASEDIR} ] || mkdir -p ${BASEDIR}
42 | [ -d ${BASEDIR} ] || fail could not create ${BASEDIR}
43 | [ -G ${BASEDIR} ] || fail ${BASEDIR} not owned by ${MYGROUP}
44 | [ -O ${BASEDIR} ] || fail ${BASEDIR} not owned by ${MYIDENT}
45 |
46 | # store current situation in a file
47 |
48 | clean_last >${CURRENT}
49 |
50 |
51 |
52 | # Is there a history file?
53 | #
54 | if [ -f ${HISTORY} ]
55 | then
56 |   # Yes - has the situation changed since last time?
57 |   #
58 |   if ! `cmp --silent $CURRENT $HISTORY`
59 |   then
60 |       # Yes - mail root about it
61 |       #
62 |       diff $HISTORY $CURRENT |mail root -s "Login report"
63 |   fi
64 | fi
65 | #
66 | # Put current situation into history file
67 | #
68 | mv ${CURRENT} ${HISTORY}
69 | [ $? -eq 0 ] || fail mv ${CURRENT} ${HISTORY}
70 | exit 0
  \-----------------------------------------------------------------
```

This script could be run by **cron**, let's say, every 5 minutes. At line 01, we see the sequences ("*hash-bang*") that tell the shell that this script should be run by /bin/sh. On most Linux systems, this will be a link to the **bash** shell. The script uses a file to maintain its state between invocations (the history file) and a workfile. The history file will contain the login information as it was written at the previous invocation of the script and the workfile will contain the most recent login information. The locations for these files are specified on lines 01-12. Using variables to configure a script is both a common and time-honored practice. That way, you only need to change these variables should you ever desire another location for these files, instead of having to change the names throughout the entire script.

On lines 13-20, a rudimentary fail-function is defined. It prints its arguments and exits, passing the value 1 to the calling shell. Passing a non-zero value to the calling program is, by convention, a signal that some error occurred whilst executing the program. In the rest of the script, we can use the construct:

fail {the text to print before exiting}

which will print:

Failed: {the text to print before exiting}

and will abort the script. If the script is run by **cron**, any (error)output will be mailed by default to the owner of the entry.

Lines 22-34 contain a function that serves as a wrapper around the **last** command. Its output is filtered: it removes lines that start with reboot (line 29), empty lines (line 30) and the line that reports when the wtmp file was created (line 31). All other lines are sent through unchanged.

In lines 36 and 37, the identity of the user running the script is determined. These values used to create readable reports on failure. Lines 39-44 check the environment. Line 41 checks if the base directory exists and, if not, tries to create it. Line 42 rechecks the existence of the directory and exits if it (still) is not there. Lines 43 and 44 check whether or not the base-directory is owned by the current user and group. If one of these tests fails, an error messages is created by the fail function and the script is aborted.

Lines 48 and 49 execute our special filtered version of **last** and send the output to our workfile. That file now contains the most current log information. Lines 52-64 checks whether a history file already exists. This will always be the case, unless this is the first invocation of the script. On line 58, a comparison between the old and new database is made. If there is no difference, nothing is done. Otherwise, the command **diff** is used to mail the differences to root. Finally, in line 68, the history file is overwritten by the current database.

The aforementioned method has plenty of flaws: not every program logs to wtmp, and the

reports will be delayed for, at most, the length of the interval you specified to run the checkscript. Also, the output format of diff is not the most user-friendly output. You may find it a rewarding operation to refine this script yourself.

---

[25] that value can be changed by specifying a command-line flag to **syslogd** on startup.

[26] Of course, your sum will almost certainly differ from this example, unless you have somehow gotten hold of a copy of my logfile...

Chapter 14. Troubleshooting (2.214)

Prev Next

Chapter 14. Troubleshooting (2.214)

Revision: $Revision: 1.6 $ ($Date: 2007/01/18 14:45:53 $)

This topic has a total weight of 7 points and contains the following objectives:

Objective 2.214.1; (Not Implemented by LPI)

This objective was not defined by LPI.

Objective 2.214.2 Creating recovery disks (1 point)

Candidate should be able to: create both a standard bootdisk for system entrance, and a recovery disk for system repair.

Objective 2.214.3; Identifying boot stages (1 point)

Candidate should be able to: determine, from bootup text, the 4 stages of boot sequence and distinguish between each.

Objective 2.214.4; Troubleshooting LILO (1 point)

Candidate should be able to: determine specific stage failures and corrective techniques.

Objective 2.214.5; General troubleshooting (1 point)

A candidate should be able to recognize and identify boot loader and kernel specific stages and utilize kernel boot messages to diagnose kernel errors. This objective includes being able to identify and correct common hardware issues, and be able to determine if the problem is hardware or software.

Objective 2.214.6; Troubleshooting system resources (1 point)

A candidate should be able to identify, diagnose and repair local system environment.

Objective 2.214.7; Troubleshooting network issues (1 point)

A candidates should be able to identify and correct common network setup issues to include knowledge of locations for basic configuration files and commands.

Objective 2.214.8; Troubleshooting environment configurations (1 point)

A candidate should be able to identify common local system and user environment configuration issues and common repair techniques.

Creating recovery disks (2.214.2)

Revision: $Revision: 1.8 $

Candidate should be able to: create both a standard bootdisk for system entrance, and a recovery disk for system repair.

Key files, terms and utilities include:

/usr/sbin/rdev
/bin/cat
/bin/mount(includes -o loop switch)
Any standard editor
/sbin/lilo
/bin/dd
/sbin/mke2fs
/etc/fstab, /etc/inittab
/usr/sbin/chroot
Familiarity with the location and contents of the LDP Bootdisk-HOWTO *(http://www.ibiblio.org/pub/Linux/docs/HOWTO/Bootdisk-HOWTO)*

Resources: BootFAQ; the **man** pages for the various commands.

Why we need bootdisks

A bootdisk is a small Linux system on a diskette. You can boot a kernel from it, hence the name. The kernel will mount a root filesystem, which typically is either stored on the same diskette or on a separate *root disk*. That filesystem contains a number of necessary files and devices, and a set of software tailored for a specific task. You can expect basic commands like /bin/mount, /bin/sh or /bin/bash to be in the bootdisk's root file system and basic configuration files like /etc/inittab - just enough to satisfy the needs the bootdisk was intended for.

The software found on a bootdisk is mostly used to perform a subset of system maintenance functions. Often, the boot disk's root filesystem contains software to recover from system errors, for example to enable you to inspect and repair a damaged system. Sometimes they are used to perform a very specific task, for example to enable easy software upgrades for large quantities of MS-Windows laptops[27].

Since there is just a limited amount of data you can store on a diskette, often compression

techniques are used to create a compressed image of the filesystem on disk. In these cases the kernel will decompress the image into memory and subsequently mount it. The kernel needs to be built with RAM-disk support to enable it to do this. In some cases the available space is insufficient for both the kernel and its root filesystem. In these cases, the kernel is put on one disk (the *boot* disk) and the (compressed) root filesystem on another (the *root* disk). Sometimes one or more *utility disks* are used to store additional data, for example additional utilities that do not fit on your root disk.

The *Bootdisk-HOWTO* contains an excellent and very detailed description on how to build various types of boot and root disks. In the sections below we give an overview of the material presented there. Please refer to the HOWTO for additional information.

Create a bootdisk

To build a bootdisk, you will need to ..

- select or build a kernel
- copy the kernel to a floppy disk either by:
  - using LILO or
  - using dd

- create and populate a root filesystem
- decide whether or not to use a separate disk for the root filesystem
- compress the root filesystem
- copy the root filesystem to floppy disk
- set the ramdisk word

Each of these steps is outlined below. More specific details can be looked up in the FAQ.

The boot disk typically contains the kernel to boot. Hence, you first need to determine the functionality your boot-kernel will need and either build a new kernel or select one that was built before and matches the specifications. For example, to create a bootdisk to be used for checking filesystems, you need to have a kernel that recognizes the proper filesystems. However, it does not necessarily need networking functionality. In some cases it suffices to copy over your existing kernel, but in many cases your default (compressed) kernelimage will be too bloated to fit on a diskette, especially if you aim to build a combined boot/root disk. The kernel building process is described in more detail in the section called "Kernel Components (2.201.1)" and on.

After selecting/building the kernel you need to copy it on to the boot disk. You can either use **LILO** or copy over the kernel using **dd**.

- The method which uses **LILO** consumes more disk space, but adds some flexibility, since you will be able to specify extra parameters during boot. You will need to create a small filesystem on the boot disk for **LILO**, just big enough to contain the kernel

and 50 blocks worth of additional space for inodes. You then create a filesystem on the disk, using **mke2fs**, and fill it with some device- and configuration files, see the FAQ for details. Then you run **lilo** on it.

- The other method uses **dd** to copy over the kernelimage to the disk. In this case you do not need to make a filesystem, but need to configure the kernel itself to instruct it where to find its rootdevice. This can be done using the **rdev** command.

In both cases next you need to set a value in the so-called *ramdisk word* in the kernel. This is a well defined location in the kernel binary that is used to specify where the root filesystem is to be found, along with other options. The word consists of 16 bits. The first 11 bits are used to set the offset (specified in 1024 byte blocks) where on the disk the root filesystem image will be stored. If bit 15 is set the kernel will prompt you for a diskette before trying to mount the root filesystem. This enables you to use a separate root disk. Bit 14 is used to signify that a RAMdisk should be used. You will need to calculate the value for the RAMDISK word by hand and set its value via the **rdev** command.

The creation of a root filesystem is by far the most complex part of the job. It requires setting up a partition somewhere (a physical partition, a file mounted via the loopback interface or a ramdisk), zeroing it out to remove random garbage (which allows for optimal compression later), creating a filesystem on it, mounting it, populating it with the files you want, unmounting it, compressing it and finally copying the compressed image onto a floppy. You will need to ensure the consistency of the filesystem contents, for example make sure that all (dynamic) libraries that are needed are available and all necessary devices are created. All of these steps are described in more detail in the FAQ.

Big files have little files upon their back..

Linux makes it fairly easy to create a filesystem within/on a *file* instead of on a disk or ramdisk: it uses *loopback devices*. You could create an iso9660 CD ROM image (typically using the **mkisofs** command) and mount it without having to burn it to CD first by using a loopback device. A less commonly known technique uses a "plain" file. To make this work you need to create an zeroed out file first:

```
dd if=/dev/zero of=fsfile bs=1k count=100
```

This creates a file that contains 100K of zero bytes, named "fsfile". Next, you need to associate one of the loopback devices with that file. This is done using the **losetup** command:

```
losetup -e none /dev/loop0 fsfile
```

The `-e none` is default and specifies that encryption should not be used in this case (check out the manual page for **losetup(8)** for more details). Now, you can make a filesystem on it, in this case an **ext2** filesystem:

```
mkfs -t ext2 /dev/loop0 100
```

Next, mount this newly created filesystem somewhere in your hierarchy, let's say on /mnt:

```
mount -t ext2 /dev/loop0 /mnt
```

You can use this filesystem any way you want to. To unmount it, use:

```
umount /dev/loop0
losetup -d /dev/loop0
```

There are various software packages available to aid the creation of boot- and rescuedisks. Using these packages you can just specify a set of files for inclusion and the software automates (to varying degrees) the creation of a bootdisk. Some of these packages enable creation of very compact, yet rich "micro-distributions" that contain hundreds of commands and a kernel on one diskette.

initrd

Newer kernels (as of 2.0) allow a kernel to boot in two phases. Initially, the kernel can load an initial ramdisk image (hence the name **initrd**) from the boot disk, which contains a root filesystem and a program (/linuxrc) to be run. linuxrc can perform various (interactive) tasks, e.g. ask the user for additional configuration options. It often is used to add additional modules in a bare-bone kernel. After this initial program exits, the kernel continues loading the real root image and booting continues normally.

Create a recovery disk

A recovery (or rescue-) disk is in fact just a boot/root disk and is created likewise (the section called "Why we need bootdisks"). But besides the generic software that makes up a bootdisk, it also contains information specific for your distribution or system. It is used to recover from fatal system crashes and can be used to restore your systems functionality, for example by providing software to restore backups (**cpio**, **tar**). It usually contains information about your system like partition information and a kernel specific to your systems hardware. Most Linux distributions offer you the option to create a rescue disk during system installation.

---

[27] This actually occurred a few years ago, when a large company needed to upgrade the OS on hundreds of MS-Windows based laptops. A special *Linux* boot disk was created, which contained stripped down versions of Linux, the PCMCIA tools and the Samba server. By booting that disk in a networked laptop it became a sambaserver that had the harddisk of

the laptop mounted, ready for updates.

Prev Next
Monitoring your system Home Identifying boot stages (2.214.3)

Identifying boot stages (2.214.3)

Revision: $Revision: 1.7 $

Candidate should be able to: determine, from bootup text, the 4 stages of boot sequence and distinguish between each.

Key files, terms and utilities include:

boot loader start and hand off to kernel
kernel loading
hardware initialization and setup
daemon initialization and setup

Resources: the **man** pages for the various commands.

The bootstrap process

The boot process has been described at length in Chapter 2, *System Startup (2.202)*. This section underlines and enhances that description. We will limit our discussion to PC hardware, though most other hardware uses similar schemes.

The PC boot process is started on powerup. The processor will start execution of code contained in the *Basic In- and Output System* (BIOS). The BIOS is a program stored in Read Only Memory (ROM) and is a part of the PC hardware. Apart from the bootstrap code it contains routines to set up your hardware and to communicate with it. Most of the code in the BIOS is never used by Linux, but the bootstrap code is.

The bootstrap code will load a block of data from sector 0, cylinder 0 of what has been configured to be your boot drive. In most cases this will be the first floppy drive. If reading the floppy disk fails or no floppy disk was inserted, the program in the BIOS will try to load the first sector from the first (specified) hard disk. Most BIOSes allow you to set up an alternate order, i.e. to try the hard disk first or first try to boot from CD.

Wherever the data was found (and *if* it was found, of course) the BIOS will load it into memory and try to execute it as if it were a program. In most cases the data either consists of code from a boot loader such as LILO, or the start of an operating system kernel like Linux. If the code on the boot sector is illegible or invalid, the BIOS will try the next

bootdevice.

Kernel loading

As can be determined from the text above, there are two ways for the kernel to be loaded:

- by using the kernelcode itself. The first sector of the boot disk will contain the first sector of the Linux kernel itself. That code loads the rest of the kernel from the boot device.
- by using a bootstrap loader. There are 2 well-known bootstrap loaders for Linux: GRUB (GRand Unified Bootloader) and LILO. LILO has by far the largest installed base, formally GRUB is still beta software. However, GRUB has a number of advantages over LILO, such as built in knowledge of filesystems. Hence GRUB is capable of loading configuration files and the kernel directly by their filename. LILO uses a different method: the physical location (track/sector/offset) for the kernel is stored at installation time. The bootloader part of LILO doesn't need knowledge of the filesystem it is booting from. It is the most commonly used bootloader. It is the only bootstraploader you need to know to pass the LPIC-2 exam, but it never hurts to be aware of an alternative.

If LILO has been used, it will locate the kernel, load it and execute it. If the kernel has been raw-copied to a diskette its first sector also contains code that loads the rest of the kernelcode from the boot device and consequently executes it.

The kernel will initialize its internal data structures and device drivers. Once it is completely initialized, it consults the contents of the *ramdisk word*, a fixed address in its binary that specifies where the kernel can find the filesystem that will be mounted as root (`/', the root filesystem). The ramdisk word also can specify that the filesystem is a RAMdisk. A RAMdisk is a memory region that is loaded with a (optionally compressed) image of a filesystem, and that is used if it were a hard disk. If the kernel can not find the root filesystem it halts.

Daemon initialization

Assuming all went well the kernel now is up and running and has mounted its root filesystem. Next, the kernel Will start up the **init** program, located in either /bin or /sbin. **init** uses the configuration file /etc/inittab to determine which program(s) to start next.

The way **init** is used to start up the initial processes varies from distribution to distribution. **init** can be configured in many ways. But in all cases a number of commands will be issued to set up the basic system such as running **fsck** on hard disks, initializing swapping and mount disks that are configured in /etc/fstab. Next, a group of commands (often scripts) are executed. They define a so called *runlevel*. Such runlevels define a set of processes that need to be run to get the system in a certain state, for example multi-user mode or single-user mode.

The initdefault entry in the /etc/inittab file defines the initial runlevel of the system. If there is

no such entry or the configuration file was not found **init** will prompt for a runlevel at the system console. Consequently, all processes specified for that runlevel in the inittab file will be started. In some cases the initial scripts are specified by the sysinit label in the **init**, in other cases they are considered part of a runlevel.

When a runlevel defines multi-user use, typically a number of daemons is started next. This is done by using start-up scripts, that can be in various location, depending on the distribution you use. Typically such start-up scripts are located in the /etc/rc.d/ directory and named aptly after the software they run, e.g. **sendmail**, **inetd** or **sshd**. Typically, these scripts are linked to another level of directories, one per runlevel. In all runlevels one or more **getty** programs will be spawned, to enable user logins.

Recognizing the four stages during boot

From what was written before you now should be able to identify the four stages of the bootsequence:

- boot loader start and hand off to kernel - typically you can recognize this stage because LILO displays the four letters "L", "I", "L", and "O". Each of these letters identifies a certain stage in the initial bootprocess. Their meaning is described in more detail in the section called "LILO errors";
- kernel loading - this stage can be recognized since the kernel will display various messages, starting with the message "Loading " followed by the name of your kernel, e.g. Linux-2.2.20.
- hardware initialization and setup - can be identified by various messages that inform you about the various hardware components that were found and initialized.
- daemon initialization and setup - this is fairly distribution specific, but this stage can be recognized by messages that typically contain lines like "Starting the ... daemon".

The kernel stores its messages is a ring buffer. On most Linux systems that ring buffer is flushed to a file during the last phase of the boot process for later inspection. The command **dmesg** will display the contents of the current buffer (and actually often is used to flush the ring buffer to a file during the last phase of the boot sequence). Check the manual page for more information.

Troubleshooting LILO (2.214.4)

Revision: $Revision: 1.6 $

Candidate should be able to: determine specific stage failures and corrective techniques.

Key files, terms and utilities include:

Know meaning of L, LI, LIL, LILO, and scrolling 010101 errors
Know the different LILO install locations, MBR, /dev/fd0, or primary/extended partition
/boot/boot.b
Know significance of /boot/boot.### files

Resources: the **man** pages for the various commands, Wirzenius98, Yap98.

Booting from CD-ROM and networks

As of this writing most BIOS's let you choose booting from hard disk, floppy, network or CDROM. To give an oversight these alternatives are outlined below. Since most systems boot from hard disk, this process was described in more detail and is elaborated on later on.

Booting from CDROM requires that your hardware support the "El Torito" standard. El Torito is a specification that says how a CDROM should be formatted such that you can directly boot from it. A bootable CDROM contains contains a floppy-disk image in its initial sectors. This image is treated like a floppy by the BIOS and booted from.

Booting from the network is done using the *Boot Protocol* (BOOTP) or the *Dynamic Host Configuration Protocol* (DHCP). DHCP actually is an evolution of BOOTP. In most cases the client has no means to address the bootserver directly, so the client broadcasts an UDP packet over the network. Any bootserver that has information about the client stored will answer. If more than one server responds, the client will select one of them. Since the requesting client does not yet have a valid IP address, the unique hardware (MAC) address of its network card is used to identify it to the BOOTP server(s) in your network. The BOOTP server(s) will issue the IP address, a hostname, the address of the server where the image of the kernel to boot can be found and the name of that image. The client configures its network accordingly and downloads the specified image from the server that was specified using the *Trivial File Transfer Protocol* (TFTP). TFTP is often considered to be an unsafe protocol, since there is no authentication. It uses the UDP protocol. However, its triviality also compact implementations that can be stored in a boot-ROM, for example a PC BIOS.

After the kernel image has been retrieved, is will be started the usual way. Often, the root filesystem is located on another server too, and NFS is used to mount it. This requires a Linux kernel that allows the root filesystem to be NFS.

Booting from disk or partition

booting from floppy or disk is the common case. In previous chapters we already described the boot process used to boot from floppy. However, there is is a slight difference between floppy and hard disk boots. Both contain a *bootsector*, located at cylinder 0, head 0, sector 1. On a floppy the boot sector often contains just the boot code to be loaded in memory and executed.

booting from hard disk requires some additional functionality: a hard disk can contain one or more *partitions*, in which case the boot program needs to find out from which partition to boot. A partition in turn will contain its own bootcode sector. The sector located at cylinder 0, head 0, sector 1 is called the *master boot record* (MBR).

Information about hard disk partitions is typically stored in *partition tables*, which are data-structures stored on a special *partition sector*. There are various types of partition tables, for example IRIX/SGI, Sun or DOS. It depends on the hardware in use which type of partition table is used. In this book we focus on the classical PC (DOS) partition table, which is typical for PC hardware. By default Linux accepts and uses DOS partition tables. Support for other partition types can be enabled in the kernel. On PC hardware the partition table is part of the MBR. You can use the **fdisk** command to print out your current partition table or to create a new one.

On a PC the BIOS starts loading the first 446 bytes of cylinder 0, head 0, sector 1 into memory. These bytes comprise the boot program. That boot program is executed next. It is up to you which program to use to boot your system. By writing your own boot program you could continue the boot process any way you want. But there are many fine boot programs available, for example the DOS loader and the Linux Loader (LILO). Alternately, you can use another boot loader program, for example GRUB. Sometimes a Windows boot loader is used (i.e. Bootmagic), or even the old fashioned DOS boot loader.

DOS for example uses a loader programs that scans the partition table for a bootable partition. When an entry marked "active" was found the first sector of that partition is loaded into memory and executed. That code in turn continues the loading of the operating system.

Linux can install a loader program too. Often this will be LILO, the *Linux Loader*. LILO uses a two-stage approach: the boot sector has a boot program, that loads a boot file, the *second stage boot program*. That program presents you with a simple menu-like interface, which either prompts you for the operating system to load or optionally times out and loads the the default system. Note, that the code in the MBR is limited: it does not have any knowledge about concepts like "filesystems" let alone "filenames". It can access the hard disk, but needs the BIOS to do so. And the BIOS is not capable of understanding anything

but CHS (Cylinder/Heads/Sectors). Hence, to find its boot program, the code in the MBR needs exact specification of the CHS to use to find it. These specifications are figured out by /sbin/lilo, when it installs the boot sector.

The second stage LILO boot program needs information about the following items:

- where /boot/boot.b can be found; it contains the second stage boot program. The second stage program will be loaded by the initial boot program in the MBR;
- the /boot/map file, which contains information about the location of kernels, boot sectors etc.; this information is used mostly by the second stage boot program; see below for a more detailed description of the map file;
- the location of kernel(s) you want to be able to boot
- the boot sectors of all operating systems it boots
- the location of the startup message, if one has been defined

Remember, to be able to access these files, the BIOS needs the CHS (Cylinder/Head/Sector) information to load the proper block. This also holds true for the code in the second stage loader. LILO therefore needs a so called *map file*, that maps filenames into CHS values. This file contains information for all files that LILO needs to know of during boot, for example locations of the kernel(s), the command line to execute on boot, and more. The default name for the map file is /boot/map. /sbin/lilo uses the file /etc/lilo.conf to determine what files to map and what bootprogram to use and creates a mapfile accordingly.

More about partitions tables

The DOS partition table is embedded in the MBR at cylinder 0, head 0, sector 1, at offset 447 (0x1BF) and on. There are four entries in a DOS partition table. Only one of them can be marked as active: the boot program normally will load the first sector of the active partition in memory and deliver control to it.

An entry in the partition table contains 16 bytes, as shown in the following figure:

Figure 14.1. A (DOS) partition table entry

```
|boot? ||start              ||type  ||partition          |
|     ||cyl     |head   |sect||     ||cyl     |head   |sect|
|------||--------||------||----||------||--------||------||----|


|start in LBA               ||size in sectors            |
|------||------||------||------||------||------||------||------|
```

As you can see, each partition entry contains the start and end location of the partition specified as the Cylinder/Head/Sector of the hard disk. Note, that the "Cylinder" field has 10

bits, therefore the maximum number of sectors that can be specified is (2^10==) 1024. BIOSes traditionally use CHS specifications hence older BIOSes are not capable of accessing data stored beyond the first 1024 cylinders of the disk.

As disks grew in size the partition/disk sizes could not be properly expressed using the limited capacity of the CHS fields anymore. An alternate method of addressing blocks on a hard disk was introduced: *Logical Block Addressing* (LBA). LBA addressing specifies sections of the disk by their *block number* relative to 0. A block can be seen as a 512 byte sector. The last 64 bits in a partition table entry contain the "begin" and "end" of that partition specified as LBA address of the begin of the partition and the number of sectors.

Tip

Remember that your computer boots using the BIOS disk access routines. Hence, if your BIOS does not cope with LBA addressing you may not be able to boot from partitions beyond the 1024 cylinder boundary. For this reason people with large disks often create a small partition somewhere within the 1024 cylinder boundary, usually mounted on /boot and put the boot program and kernel in there, so BIOS can boot Linux from hard disk. Once loaded, Linux ignores the BIOS - it has its own disk access procedures which are capable of handling huge disks.

The "type" field contains the type of the partition, which usually relates to the purpose the partition was intended for. To give an impression of the various types of partitions available, a screen dump of the "L"ist command within **fdisk** follows:

```
 0  Empty           17  Hidden HPFS/NTF 5c  Priam Edisk     a6  OpenBSD
 1  FAT12           18  AST Windows swa 61  SpeedStor       a7  NeXTSTEP
 2  XENIX root      1b  Hidden Win95 FA 63  GNU HURD or Sys b7  BSDI fs
 3  XENIX usr       1c  Hidden Win95 FA 64  Novell Netware  b8  BSDI swap
 4  FAT16 <32M      1e  Hidden Win95 FA 65  Novell Netware  c1  DRDOS/sec (FAT-
 5  Extended        24  NEC DOS         70  DiskSecure Mult c4  DRDOS/sec (FAT-
 6  FAT16           3c  PartitionMagic  75  PC/IX           c6  DRDOS/sec (FAT-
 7  HPFS/NTFS       40  Venix 80286     80  Old Minix       c7  Syrinx
 8  AIX             41  PPC PReP Boot   81  Minix / old Lin db  CP/M / CTOS / .
 9  AIX bootable    42  SFS             82  Linux swap      e1  DOS access
 a  OS/2 Boot Manag 4d  QNX4.x          83  Linux           e3  DOS R/O
 b  Win95 FAT32     4e  QNX4.x 2nd part 84  OS/2 hidden C:  e4  SpeedStor
 c  Win95 FAT32 (LB 4f  QNX4.x 3rd part 85  Linux extended  eb  BeOS fs
 e  Win95 FAT16 (LB 50  OnTrack DM      86  NTFS volume set f1  SpeedStor
 f  Win95 Ext'd (LB 51  OnTrack DM6 Aux 87  NTFS volume set f4  SpeedStor
10  OPUS            52  CP/M            93  Amoeba          f2  DOS secondary
11  Hidden FAT12    53  OnTrack DM6 Aux 94  Amoeba BBT      fd  Linux raid auto
12  Compaq diagnost 54  OnTrackDM6      a0  IBM Thinkpad hi fe  LANstep
14  Hidden FAT16 <3 55  EZ-Drive        a5  BSD/386         ff  BBT
```

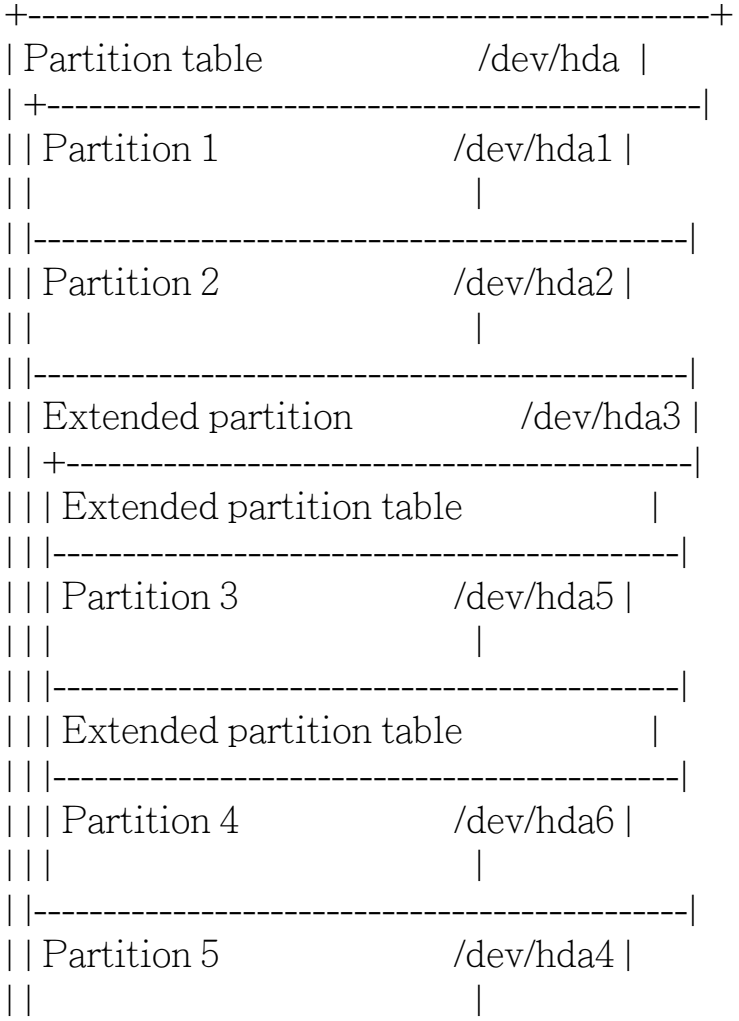
16 Hidden FAT16 56 Golden Bow

Extended partitions

The design limitation that imposes a maximum of four partitions proved to be troublesome as disks grew larger and larger. Therefore, a work-around was invented: by specifying one of the partitions as a "DOS Extended partition" it in effect becomes a container for more partitions aptly named *logical partitions*. The "Extended partition" can be regarded as a container, that holds one or more logical partitions. The total size of all logical partitions within the extended partition can never exceed the size of that extended partition.

In principle Linux lets you create as many logical partitions as you want, of course restricted by the physical boundaries of the extended partition and hardware limitations. The logical partitions are described in a linked list of sectors. The four primary partitions, present or not, get numbers 1-4. Logical partitions start numbering from 5. The main disk contains a partition table that describes the partitions, the extended partitions contain logical partitions that in turn contain a partition table that describes a logical partition and a pointer to the next logical partitions partition table, see the ASCII art below:

Figure 14.2. Partition table setup



+---+

The LILO install locations

LILO's first stage loader program can either be put in the MBR, or it can be put in any partitions boot sector. Of course, you could put it in both locations if you wanted to, for example in the MBR to decide whether to boot Windows, DOS or Linux and if Linux is booted, its boot sector could contain LILO's primary loader too, which would for example enable you to choose between different versions/configurations of the kernel.

The tandem "Linux and Windows" is frequently used to ease the migration of services to the Linux platform or to enable both Linux and Windows to run on the same computer. To dual boot Linux and Windows 95/98, you can install LILO on the master boot record. Windows NT and Windows 2000 require their own loader in the MBR. In these case, you can install LILO in the Linux partition as a secondary boot loader. The initial boot will be done by the Windows loader in the MBR, which then can transfer control to LILO.

LILO backup files

/sbin/lilo can create the bootprogram in the MBR or in the first sectors of a partition. The bootprogram, sometimes referred to as *the first stage loader* will try to load the *second stage boot loader*. The seconds stage bootloader is contained in a file on the boot partition of your Linux system, by default it is in the file /boot/boot.b.

If you use /sbin/lilo to write the bootprogram it will try to make a backup copy of the old contents of the bootsector and will write the old contents in a file named /boot/boot.####. The hash symbols are actually replaced by the major and minor numbers of the device where the original bootsector used to be, for example, the backup copy of the MBR on the first IDE disk would be stored as /boot/boot.0300: 3 is the major number for the device file /dev/hda, and 0 is the minor number for it. /sbin/lilo will not overwrite an already existing backup file.

LILO errors

When LILO loads itself, it displays the word

LILO

Each letter is printed before or after performing some specific action. If LILO fails at some point, the letters printed so far can be used to identify the problem.

(nothing)

No part of **LILO** has been loaded. Either **LILO** isn't installed or the partition on which its

boot sector is located isn't active.

L error

The first stage boot loader has been loaded and started, but it can't load the second stage boot loader. The two-digit error codes indicate the type of problem. This condition usually indicates a media failure or a geometry mismatch. The most frequent causes for a geometry mismatch are not physical defects or invalid partition tables but errors during the installation of **LILO**. Often these are caused by ignoring the 1024 cylinder boundary.

This error code signals a transient problem - in that case LILO will try to resume or halt the system. However, sometimes the error code is not transient and LILO will repeat it, over and over again. This means that you end up with a scrolling screen that contains just the error codes. For example: the error code "01" signifies an "illegal command". This signifies that the disk type is not supported by your BIOS or that the geometry can not correctly be determined. Other error codes are described in full in the LILO's user documentation.

LI

The first stage boot loader was able to load the second stage boot loader, but has failed to execute it. This can either be caused by a geometry mismatch or by moving /boot/boot.b without running the map installer.

LIL

The second stage boot loader has been started, but it can't load the descriptor table from the map file. This is typically caused by a media failure or by a geometry mismatch.

LIL?

The second stage boot loader has been loaded at an incorrect address. This is typically caused by a subtle geometry mismatch or by moving /boot/boot.b without running the map installer.

LIL-

The descriptor table is corrupt. This can either be caused by a geometry mismatch or by moving /boot/map without running the map installer.

LILO

All parts of **LILO** have been successfully loaded.

Prev | Up | Next
Identifying boot stages (2.214.3) | Home | General troubleshooting (2.214.5)

General troubleshooting (2.214.5)

Revision: $Revision: 1.10 $

A candidate should be able to recognize and identify boot loader and kernel specific stages and utilize kernel boot messages to diagnose kernel errors. This objective includes being able to identify and correct common hardware issues, and be able to determine if the problem is hardware or software.

Key files, terms and utilities include:

screen output during bootup
/bin/dmesg
kernel syslog entries in system logs (if entry is able to be gained)
various system and daemon log files in /var/log/
/sbin/lspci
/bin/dmesg
/usr/bin/lsdev
/sbin/lsmod
/sbin/modprobe
/sbin/insmod
/bin/uname
location of system kernel and attending modules /, /boot, and /lib/modules
/proc filesystem
strace
strings
ltrace
lsof

Resources: the **man** pages for the various commands, Vasudevan02.

A word of caution

Debugging boot problems (or any other problems for that matter) can be complex at times.

Your best teacher is experience. However, by carefully studying the boot messages and have the proper understanding of the mechanisms at hand you should be able to solve most, if not all, common Linux problems.

You need a good working knowledge of the boot process to be able to solve boot problems. In previous sections the boot process was described in detail as was system initialization. By now you should be able to determine in what stage the boot process is from the messages displayed on the screen. You should be able to utilize kernel boot messages to diagnose kernel errors. If not, please re-read the previous sections carefully. We will provide a number of suggestions on how to solve common problems in the next sections. However, it is beyond the scope of this book to cover all possible permutations. If your problem is not covered here, search the web for peers with the same problem, consult your colleagues, read more documentation, investigate and experiment.

Cost effectiveness

In an ideal world you always have plenty of time to find out the exact nature of the problem and consequently solve it. In the real world you will have to be aware of cost effectiveness. Look for the most (cost-) effective way to solve the problem. For example: let's say that initial investigation indicates that the boot disk has hardware problems. You do not know yet the exact nature of the problems, but have eliminated the most common causes. The disk still does not work reliably. Hence, you need more time to investigate further. If your customer has a recent backup and deliberation learns that he agrees to go back to that situation, you might consider suggesting installation of a brand new disk and restoring the most recent backup instead. The total costs of your time so far and the new hardware are probably less than the costs you have to make to investigate any further - even more so if you consider the probability that you need to replace the disk after all.

Getting help

This book is not an omniscient encyclopedia that describes solutions for all problems you may encounter. If you get stuck, there are many ways to get help. First, you could call or mail the distributor of your Linux version. Often you are granted installation support or 30 day generic support. It may be worth your money to subscribe to a support network - most distributions offer such a service at nominal fees.

Some distributions grant on-line (Internet) access to support databases. In these databases you can find a wealth of information on hardware and software problems. You can often search for keywords or error messages and most of them are cross-referenced to enable you to find related topics. Some URLs you may check follow:

SuSE

http://www.suse.com/us/business/services/support/index.html

Red Hat

*http://www.redhat.com/apps/redirect.apm/services/consulting/?rhpage=/index.html/ent_services***

Debian

http://www.debian.org/doc/FAQ/ch-support.html

Mandrake

http://www.mandrakeexpert.com/index1.php

You can also **grep** or **zgrep** for (parts of) an error message or keyword in the documentation on your system, typically in and under /usr/doc/, /usr/share/doc/ or /usr/src/linux/Documentation. Additionally, you could try to enter the error message in an Internet search engine, for example *http://www.google.org*. This often returns URLS to FAQ's, HOW-TO's and other documents that may contain clues on how to solve your problem. Usenet news archives, like *http://www.deja.com* are another resource to use.

Generic issues with hardware problems

Often, Linux refuses to boot due to hardware problems. If a system boots and seems to be working fine, it still may have a hardware problem. If it is not working with Linux, but is working fine with other operating systems, for example DOS or Windows, this too often signifies hardware problems. Linux assumes the hardware to be up to specifications and will try to use it to its (specified) limits.

In general: regularly check the system log files for write and read errors. They indicate that the hardware is slowly becoming less reliable. Other indications of lurking hardware problems are: problems when accessing the CDROM (halt, long delays, bus errors, segmentation faults), kernel generation or compilation of other programs aborts with signal 11 or signal 7, scrambled or incorrect file contents, memory access errors, graphics that are not displayed correctly, CRC errors when accessing the floppy disk drive, crashes or halts during boot and errors when creating a filesystem.

To discover lurking hardware errors, you can use a simple, yet effective test: create a small script that compiles the kernel in an endless loop, for example:

```
#
# adapted from http://www.bitwizard.nl/sig11
#
cd /usr/src/linux
#
c=0
while true
```

```
do
  make clean &> /dev/null
  make -k bzImage > log.${c} 2> /dev/null
  c=`expr ${c} + 1`
done
```

Every iteration of this loop should create a log file - all log files should have the exact same content. You could use **sum** or **md5sum** to check this. If you detect differences between the log files this often is an indication some hardware problem exists.

Resolving initial boot problems

If your system does not boot (anymore) you want to retrieve basic system functionality: your system should boot and the kernel should load. After that, you often are able to resolve the other problems using the rich set of debugging tools Linux offers.

There are a number of common boot problems. One group relates to the MBR and bootstrap files. Data corruption or accidental deletion of files or the boot partition will prevent your drive from booting. Another group clearly relates to hardware failures on the boot-drive.

hardware problems. If a hardware component failure causes the system to refuse to boot you typically get one of the following clues: the BIOS reports errors like "No Fixed Disk Found" or "Disk Controller Error". Sometimes, numerical messages are displayed, often in the 1700 range, e.g. "1701", "1791" etc. If the controller is in error, a message that indicates this is often shown. In the most obvious case, the disk does not even start spinning.

Start by ensuring that your BIOS was set up correctly. The disk geometry needs to be set correctly to enable your BIOS to see the drive. Often, a Plug And Play option can be set in the BIOS. Set the option so that PNP is deactivated. If an additional option "Reset Configuration Data" or "Update ESCD" is offered, please set it to "yes" or alternatively to "enabled".

If you have problems with a drive you just added to the system, the problems are often caused by incorrect cabling or jumper selections. Make sure all connectors are properly seated. Ensure that IDE master and slave drives are jumpered and cabled properly. UDMA drives make use of special twisted pair cabling. Your BIOS needs to be able to access the disk during boot, so make sure your drive geometry and/or type has been correctly specified in your BIOS.

You should verify the seating of the connectors, check the cables between disk and motherboard. Check for corrosion. Also check the power cables. You can remove the cables and measure them, using a simple ohm meter or buzzer, and/or replace them with new ones. A "hard disk controller" error message can often be caused by bad cabling too. Always check your cabling first, even when the symptoms seem to indicate a controller error. If the error persists try to replace the controller card.

software problems. If a software failure causes the system to refuse to boot, you typically get one of the following clues: LILO does not display all of its four letters, LILO hangs, you get a screen with scrolling error codes, e.g. "010101..", the system report something like "drive not bootable, insert system disk", the system boots another operating system than intended.

If you are able to boot a floppy you can use a boot floppy or rescue disk to boot a kernel (the section called "Why we need bootdisks"). You could use a rescue floppy that boots a kernel that has the root filesystem set to your hard disk (using the **rdev** command). Or you could use a special tiny Linux distribution, like *tomsrtbt* (*http://www.toms.net/rb/*) and mount the root filesystem by hand. Make sure the rescue disk contains he necessary functionality/modules to fit your hardware, e.g. if you have SCSI disks, be sure the drivers are either compiled into your boot kernel or available as modules. If the boot floppy has booted you should start by checking the validity of your MBR. When the LILO bootup messages indicate a geometry error or a related error, you should verify that /sbin/lilo ran correctly. If you are in doubt, you can rerun it, provided you have access to (a backup copy of) the lilo.conf file. Also check if the error is caused by the 1024 cylinder boundary problem: your kernel(s) and related files should be accessible for your BIOS, and some BIOSes (as a rule of thumb: made before 1998) are not able to access disk cylinders above 1024.

Next, verify that the partition table in the MBR is correct by running **fdisk**. Verify that at least one partition is marked as "boot" or "active" - some BIOSes require that the Linux boot partition to be marked "bootable", some distributions may require this too. If the partition table is incorrect, you will need to repair it. If you have access to a backup of your boot systems MBR (including the partition table, hence: the first 512 bytes of your hard disk) and have put it on a floppy, now is the time to recover it. Alternately, you may have printed the partition list; if so you can use **fdisk** to restore the partition table.

If the partition table looks correct when you print it in **fdisk** the next step is to take a look at the root filesystem of your drive. You can use the **fsck** command to repair filesystem problems on your root file system. If all else fails, you have to resort to reformatting and/or repartitioning your disk and restore the latest back up or even may need to replace the disk.

Resolving kernel boot problems

If the initial boot sequence could be completed, the kernel is loaded and tries to mount its root filesystem. This can either be a RAMDISK (**initrd**) or a partition on a hard disk. Of course, the kernel needs to be able to access (all of) its memory and the filesystem on the disk, which may require certain device drivers in the kernel. When the root filesystem could be mounted additional programs can be executed and additional kernel modules may be loaded to add functionality. If the root filesystem is a RAM disk, it may contain programs to load the modules needed to address other hardware devices such as disks. In these phases you could experience module loading problems, which can result in (partial) hardware inaccessibility.

In many cases it is not clear whether or not the cause of a problem lies with the hardware or the software. However, most of these errors result from invalid configuration.

hardware problems. If a hardware problem causes the system to refuse to boot, you typically get one of the following clues: "PANIC VFS unable to mount root fs on ##:##", modprobe reports errors loading a module, IRQ/DMA conflicts can be reported, parts of your hardware do not work or intermittent errors occur. As a rule of thumb, if your kernel boots, but your system consequently hangs or issues error messages, you should check for software problems and configuration problems first. If this does not resolve the problem, check your hardware (the section called "Generic issues with hardware problems").

software problems. If a software problem causes the system to refuse to boot, you typically also get one of the clues we listed under "hardware errors": "PANIC VFS unable to mount root fs on ##:##", modprobe reports errors loading a module, IRQ/DMA conflicts can be reported.

The "PANIC VFS unable to mount..." message occurs when the kernel could be loaded, but either the ramdisk or the physical partition could not be mounted. This can be the result of forgetting to run /sbin/lilo after a kernel change or update, or forgetting to run rdev if case you put the kernel-image directly into the boot-sectors of a partition. The PANIC message is sometimes caused by inaccessibility of (parts of) your system's memory, for example when it tries to mount the RAM-disk as its root filesystem. Older kernels may require you to set the amount of RAM by rebooting and setting the boot parameter:

mem=<size-of-memory-in-Kbytes>

In order to recognize memory above 64 Meg, it may be necessary to append the "mem=" option to the kernel command line permanently. If you are using LILO for your boot loader, you would do this in the lilo.conf file. For example, if you had a machine with 128 Meg you would type:

append="mem=131072K"

To help you to determine what device the kernel tries to mount, the "PANIC" message contains the major and minor number of the device, e.g. 9:0 for /dev/md0 (the first virtual RAID disk, the section called "Software RAID") or 8:3, the first SCSI disk /dev/sda. This can pinpoint the area where you should start your search for configuration errors. For example: if this message points to the /dev/md0 device, you should check if the kernel was compiled with support for software RAID, ditto for SCSI disks.

In most cases, your should have your system booted by now. It may be that you still see numerous errors, e.g. daemons won't start, your network card or sound card does not work, In the next sections we will describe tools that aid you with troubleshooting and give hints and tips on how to resolve more problems.

Resolving IRQ/DMA conflicts

IRQ conflicts are a common source of problems. You can use **dmesg** to see which interrupts were required by the drivers and compare this with the contents of /proc/interrupts or the output of **lsdev** to determine conflicts. The IRQ a PCI device uses is also reported in /proc/pci or by using the **lspci** program. Sometimes a card could not obtain an IRQ since the BIOS assigned all of them to non-PCI (ISA) cards. Check your BIOS settings if you suspect this to be the case.

Under some conditions IRQ's can be shared between two devices. Devices on the PCI bus may share the same IRQ interrupt with other devices on the PCI bus provided the driver software supports this. In other cases where there is potential for conflict, there should be no problem if no two devices with the same IRQ are ever in use at the same time. Even if devices with conflicting IRQs are used simultaneously one of them will likely have its interrupts caught by its device driver and may work. The other device(s) will likely behave like they were configured with the wrong interrupts.

Troubleshooting tools

Linux has a rich set of tools that aid you in troubleshooting. An overview of the most commonly used commands and their functionality follows:

lspci**.** this command displays information about all PCI buses in the system and all devices connected to them. The command is described in more detail in the section called **"Querying your PCI bus"**.

dmesg**.** the kernel logs messages into a ring buffer. **dmesg** dumps the contents of the ring buffer to standard output. Often, the **dmesg** command is issued at the end of the boot sequence for example in one of the start up scripts, to dump the bootup messages into a file (e.g. boot.messages).

lsdev**.** is a front end to the /proc filesystem and prints out information about interrupts, I/O ports and dma settings. This gives an overview of which hardware uses what IO addresses, IRQ's and DMA channels and can aid in determining conflicts.

lsmod**.** a program that displays the modules currently in use by the kernel. Name, size, use count, an a list of referring modules are displayed. The information displayed is identical tot hat in /proc/modules. **lsmod** frequently is used to check if the proper modules could be loaded.

modprobe**.** a high level interface to **insmod**. Often, modules depend on each other and/or need to be loaded in a certain order. **modprobe** is used to make this more easy for system administrators. It uses a dependency file, which is created by **depmod**, to load modules in the right order from certain specified locations. The normal use of **depmod** is to include it somewhere in the rc-files in /etc/rc.d, so that the correct module dependencies will be

available immediately after booting the system. The configuration file /etc/modules.conf can be used to steer **depmod** and **modprobe**'s behavior. **modprobe** will unload all modules in a dependent chains if one of them fails to load. See also the section called "Kernel Components (2.201.1)".

insmod**.** installs a loadable module in the running kernel. It tries to do this by resolving all symbols from the kernel's exported symbol table. You can specify the (object) file name. If the file name is given without extension, **insmod** will search for the module in common default directories. These default locations can be overridden by the contents of an environment variable (MODPATH) or in the configuration file /etc/modules.conf.

uname**.** displays machine type, network hostname, OS release, OS name, OS version and processor type of the host.

/proc **filesystem.** is a direct reflection of the system in memory presented to you as files and directories. It provides an easy way to view kernel information and information about currently running processes. In Linux some commands read /proc directly to get information about the state of the system. It allows you to view statistical information, hardware information, network and host parameters, memory and performance information and lets you modify some parameters runtime by writing values in it.

strace**.** a very handy diagnostic tool. It runs a program or connects to a running process (e. g. a daemon) and intercepts the system calls which are received by it. By default it reports the name of the system call, its arguments and the return value on standard error. It is very useful in cases where you do not have access to the source code and also serves as a tool to be used to gain better understanding of the inner workings of certain programs. The program to be traced need not be recompiled for this.

ltrace**.** similar to **strace**, but instead of recording system calls it runs the specified command and intercepts and records the dynamic library calls which are called by the executed process and the signals which are received by that process. It can also intercept and print the system calls executed by the program. The program to be traced need not be recompiled for this, so you can use it on binaries for which you don't have the source handy.

strings**.** can print out strings - sequences of printable characters - that are hidden within non-text files, such as executables. Often used to check for names of environment variables and configurations files used by an executable.

fuser**.** accepts a filename and displays the PID's of processes using the specified files or filesystems. Comes in handy if you want to know which process uses a certain file, for example: if you are not able to unmount a filesystem this often is caused by a process that still uses a file on that filesystem. **fuser** can be used to find the PID(s) of the process(es). A consecutive **ps -p $PID** will name the process.

lsof**.** by default lists all open files belonging to all active processes. Since Unix uses the file

metaphor too for devices a open file can be a regular file, a directory, a block special file, a character special file, an executing text reference, a library, a stream or a network file (Internet socket, NFS file or UNIX domain socket.) The utility can be used to see which processes use which resources.

Prev | Up | Next

Troubleshooting LILO (2.214.4) | Home | Troubleshooting system resources (2.214.6)

Troubleshooting system resources (2.214.6)

Troubleshooting system resources (2.214.6)

Revision: $Revision: 1.7 $

A candidate should be able to identify, diagnose and repair local system environment.

Key files, terms and utilities include:

/etc/profile, /etc/profile.d/
Core system variables
/etc/bashrc (or other appropriate global shell configuration files)
/etc/init.d/
/etc/rc.*
/bin/ln
/bin/rm
Any editor of choice
/etc/ld.so.conf
/sbin/ldconfig
/sbin/sysctl, /etc/sysctl.conf

Core system variables

Applications, for example **bash**, often use variables. These variables are not necessarily available as *environment* variables, but often have a local scope: they are available for the process, but not for its children. On the other hand, every environmental variable set by an application is available for every child of that application. For example: within a shell you can access environment variables as if they were locally defined shell-variables. You can "promote" a shell variable to an environment variable by using the **export** builtin.

Many environment settings influence the behavior of applications. Some of them are very specific for just one application. But there are a number of standard variables that many application will use or expect to be in place. A list of the most commonly used environment variables follows:

Table 14.1. Commonly used environment variables

COLUMNS	the number of columns your screen has (typically 80)
HOME	the location of your home-directory
IFS	Input File Separator. IFS contains the list of separators that the shell recognizes. By default this is set to whitespace values (space, tab etc.). However, setting IFS to a non-whitespace character can pose a security risk. For example: IFS=o, the program executes the command /bin/show. “o” is considered the separator. Hence /bin/sh would be executed. Most modern command shells therefore ignore the value for IFS.
LOGNAME	name used to log in
MANPATH	colon delimited sequence of directories that contain (sources for) manual pages, see manpath(1)
PAGER	the program used to display files on screen (e.g. less or pg
PATH	colon delimited sequence of directories that contain executables
PS1..4	definitiion of the prompts the shell uses in interactive mode: PS1: the primary prompt string, shown when you initially start typing a command; PS2: secondary prompt string, shown on the second and consecutive lines if you type in a command that spans multiple lines; PS3: the prompt used when bash issues the select command[a]; PS4: the value is printed before each command bash displays during an execution trace (e.g. set -x).
ROWS	the number of rows your screen has (typically 24)
SHELL	the name of the current shell
SHLVL	if a shell is spawned, this variable is passed on to it. It will contain the SHLVL value of the parent shell, incremented by one.
TERM	the terminal type. Originally, this specified the type of terminal you physically had connected to the computer. Currently, virtual terminals are more often the case. This environment variable is used to select which controlling code sequences are sent to your (virtual) terminal, for example to set high intensity or clear the screen.
TMPDIR	allows programs that use the tempnam(3) function call to use the directory specified by this variable rather than the /tmp directory.
UID	the current users’ id
USER	the current users name

[a] The **select** command is rarely used. It allows you to specify a number of options which are printed as a menu. You are then prompted with the PS3 prompt and can select an option, which is passed to a list of statements for further processing. See the **bash** manual page for more information.

To see your environment from within a shell, type **env** or **set**. Of course, the value of individual variables can be printed using the **echo**.

Login shells

The file /etc/shells lists the valid login shells on your system. It requires fully specified filenames. An example file is:

```
/bin/bash
/bin/sh
/bin/tcsh
/bin/csh
/bin/ash
/bin/bsh
/bin/ksh
/bin/false
```

/bin/false is not a real shell, however, sometimes you want to add a user to your system that is not allowed to log in (e.g. wwwrun. Adding a user to your system requires giving him an entry in /etc/passwd, which in turn requires you to specify a valid login shell. Being able to specify /bin/false as "shell" prevents users to obtain a shell. The command **chsh** can be used by the user to set his preferred shell. "root" can use this command to set any user's shell. This command with the -l option will list the available shells. The -s option will allow the user to change his login shell:

```
$ chsh -s /bin/sh
```

Shell startup environment

Most of your users log in and are presented with a shell. A shell's behavior typically is controlled by setting a number of environmental variables. These are typically set at two levels: system wide and per user. To set them, a start-up file is loaded and executed. There are many shells - most of them have specific start-up files. Additionally, it depends on whether or not the shell is spawned by the **login** program or by another shell which startup files are executed. However, when shells are reported to display unexpected behavior, often something is set wrongly in one of the start up files.

Most login-shells will check for /etc/profile first. Typically, this file sets the TERM and PATH variables. The format for this file is the same as that of a (Posix) shell. It hence contains calls to external programs: in older days, the **fortune** command was often called here: it generates a random epigram. On older systems the contents of the message of the day file (/etc/motd) were displayed here too, on newer systems this is often done by the **login** program itself.

More settings can be put in (often shell-specific) startup files in the users HOME directory. Posix compliant shells will check for $HOME/.profile. **bash** will behave likewise, if invoked as **sh**. Often, on Linux systems, /bin/sh is a (symbolic) link to /bin/bash. Some distributions put calls to distribution or shell specific central startup files into the user level files. This can be confusing at times.

bash is the most commonly used shell for Linux systems. It will run /etc/profile if invoked from **login**, then look for $HOME/.bash_profile, $HOME/.bash_login and $HOME/.profile, in that order. It and reads and executes commands from the first one that exists and is readable. When a **bash** login shell exits it reads and executes commands from the file $HOME/.bash_logout if readable. A common mistake is to create a userlevel start up file as "root" - and forget to set the permissions correctly. At least the user should have execute and read permissions. Typically the permissions should be -rwxr-x--- (assuming the file is owned by the user).

If users complain that their shell does not recognize certain key strokes or reacts strangely on them, chances are that either the TERM variable does not match the (virtual) terminal type in use or the *key bindings* for the application (often a shell) are wrongly set. If the TERM setting is invalid the display will often be garbled too.

The behavior of programs that use the **readline** library to fetch input from the keyboard are influenced by key bind settings. The **readline** library supports **emacs** and **vi** key bindings by default and allows you to configure key bindings to certain functions within applications, for example **bash**. If the shell variable INPUTRC is set (often in /etc/profile) key-bindings are set by the file declared in the its value. Otherwise the key-bindings are set in the file $HOME/.inputrc for each individual user, or system wide in the file /etc/inputrc. The syntax for controlling key bindings is simple. All that is required is the name of the command or the text of a macro and a key sequence to which it should be bound, see the manual pages for readline(3). **bash** allows the current key bindings to be displayed or modified with the **bind** builtin command. The editing mode **bash** uses may be switched during interactive use by using the set -o option, e.g. set -o vi. Often, the user sets his preference in one of the startup files. A simple typo there can result in eccentric behavior of the shell (at least for that user).

The file /etc/login.defs (man login.defs(5)) defines the behavior of **login** on the system on systems that use shadow passwords. With it, you control the behavior of related programs too, for example chfn, chsh or groupadd. It is a plain text configuration file, used to set the initial PATH and other parameters, including how often a user must change passwords and what is acceptable as a password.

Many applications make use of other configuration files in the users HOME directory, for example:

Table 14.2. Commonly used configuration files in HOME

.emacs	a configuration file for the emacs that contains ELISP functions

.exrc	startup file for the vi editor
.fvwmrc	startup file for the fvwm window manager
.ssh	actually a directory, that contain files with the known hosts, identity and public key used by ssh
.twmrc	another one, for the twm window manager
.newsrc	keeps track of what newsgroups the user is subscribed to and what articles he read.
.Xdefaults	behaviors, fonts, and colors for X Windows System applications
.xinitrc	shell commands that run when the user logs into an X session
.xsession	shell commands that run when the user logs into an X session using Xdm

It's easy for a user to overwrite (clobber) existing files when they are redirecting output to a file. To prevent this from happening to them you can set the noclobber setting by adding the line

```
set -o noclobber
```

to their shell start-up file.

Editors

The Unix system engineer or system administrator can barely do his work without a proper editor. The definition of "proper editor" however is, and always has been, discussed at length. Roughly, there are two main stream editors available in the Unix world - thought strictly speaking one of them is far more than just an editor - **emacs** and **vi**.

vi - or one of its clones, for example the popular **vim** by Bram Molenaar, used on most Linux installations - is a crude, but effective editor. It supports an interface that allows you to keep your fingers on the keyboard all the time, and by design makes you use as less keys as possible. E.g. the use of arrow keys, function keys and other non standard keys is non-mandatory. This results in a steep learning curve for novice users but they gain a high return on that investment. They will be able to process texts faster and will be able to maintain an ergonomically acceptable working position. They will be able to work on even slightly defective keyboards/terminals. **vi** has limited functionality, but everything one needs to effectively and efficiently edit texts is available.

On the other side of the spectrum resides **emacs**. Strictly speaking, it is a text processing system. It makes use of ELISP programs to enhance its functionality. **emacs** also supports an interface that allows you to keep your fingers on the keyboard all the time. The amount of features and extensions **emacs** offers can initially be somewhat overwhelming, but it comes with excellent built in documentation. Additionally a lot of people wrote extensions for **emacs**. For example **evi**, which emulates **vi** within **emacs**.

Both **emacs** and **vi** are widely used, however, **vi** is much smaller in size than the complete **emacs** package. Hence, and given its longer history, **vi** is available on almost every Unix platform (including Linux) and on every Unix system. I therefore strongly suggest that novices start using **vi** for a while. It is omnipresent and will fit on a boot disk or rescue disk, with plenty of room to spare. Check out **emacs** if you need features like spell-checking or syntax highlighting. There is no reason not to use both applications, but as a system administrator you *must* know the **vi** basics, especially if you have to work on older systems. Many systems do not have **emacs** installed, most Linux systems do, however.

Setting kernel parameters

The **sysctl** command is used to modify kernel parameters at runtime. In Linux, it's a front end to the /proc filesystem, hence you should have compiled support for procfs if you want to use **sysctl** on Linux.

Typically you can read or set things like whether or not the kernel reacts to the three finger salute (**ctrl**-**alt**-**del**), the abilities of certain devices, such as your CDROM's ability to be able to close and open its tray, information about the way the kernel handles filesystem, such as the maximal number of inodes a filesystem can have, network information like whether or not the kernel should use IP forwarding, the kernel type and release and many, many more.

To see a list of all options available on your system you can issue the command

```
# sysctl -a |more
```

You will see a listing that could start like this:

```
# sysctl -a |more
sunrpc.nlm_debug = 0
sunrpc.nfsd_debug = 0
sunrpc.nfs_debug = 0
sunrpc.rpc_debug = 0
dev.cdrom.check_media = 0
dev.cdrom.lock = 1
dev.cdrom.debug = 0
dev.cdrom.autoeject = 0
dev.cdrom.autoclose = 1
dev.cdrom.info = CD-ROM information, Id: cdrom.c 3.11 2000/06/12
dev.cdrom.info =
dev.cdrom.info = drive name:          hdc
dev.cdrom.info = drive speed:         24
dev.cdrom.info = drive # of slots:     1
dev.cdrom.info = Can close tray:       1
```

```
dev.cdrom.info = Can open tray:        1
dev.cdrom.info = Can lock tray:        1
dev.cdrom.info = Can change speed:     1
dev.cdrom.info = Can select disk:      0
dev.cdrom.info = Can read multisession: 1
dev.cdrom.info = Can read MCN:         1
dev.cdrom.info = Reports media changed: 1
--More--
```

As you can see, the variable names (a.k.a. *key names*) consist of dot delimited parts. Each part of a variable name correlates to a directory under the /proc/sys hierarchy, for example /proc/sys/dev/cdrom/info corresponds to dev.cdrom.info. An alternate method to set or read the variables is by using plain old **cat**, e.g. you can either type:

```
# cat /proc/sys/dev/cdrom/info
```

.. or ..

```
# sysctl dev.cdrom.info
```

Note, however, that **sysctl** will prepend the key name by default, use the -n option to suppress this. To confuse matters more, you are also allowed to use the "/" slash as a separator with **sysctl**, hence these two alternate forms are valid too:

```
# sysctl dev/cdrom/info
..
# sysctl /dev/cdrom/info
..
```

sysctl can load configuration files and set the kernel environment accordingly. The default location for the configuration file is /etc/sysctl.conf. Its syntax is simple: it contains simple token = value pairs, along with comment lines and/or blank lines. An example followes:

```
# sysctl.conf sample
#
kernel.domainname = yellow.snow.nl
net.ipv4.ip_forward = 0
```

This example just sets the domain name and disables IP forwarding. You can use the -p option to instruct **sysctl** to read this file (or specify another one to read). Note, that it is possible to put whitespace and other (unmeant) tokens in the right hand side of the expressions in the configuration file. Also note that you can use the /proc interface to influence these variables, for example:

```
# echo "/sbin/modprobe" >/proc/sys/kernel/modprobe
```

Shared libraries

A *library* is an archive whose members are object files i.e., pre-compiled program code. A library makes it possible for a program to use common routines without the administrative overhead of maintaining source code. Additionally the objects in the library do not have to be compiled each time the program is compiled.

There are two types of libraries: *static* libraries and *shared* or *dynamic* libraries.

The term "static" refers to the fact that, when a static library is used whilst creating a program, the linker links in the relevant parts of the library *statically*, i.e., they become part of the resulting executable file. This consumes unnecessary memory and disk space since each new instance of a statically linked program loads parts of the same program-code its predecessors did. The exact same code will be on disk and in memory many times. Also, bug-fixing code in a statically linked library requires recompilation of all programs that use the older version of the statically linked program.

To improve this *shared libraries* were invented. When a program uses a *shared* library, the binary just contains a reference to the library, not the code itself. A special run time loader, ld.so (for a.out format files) or ld-linux.so (for ELF format files) finds the library at run-time and loads it into memory. Some additional naming conventions are used to allow real-time updates for shared libraries and support for non-backward-compatible versions. Understanding these conventions is key to your ability to maintain a Linux system.

Shared object version numbering

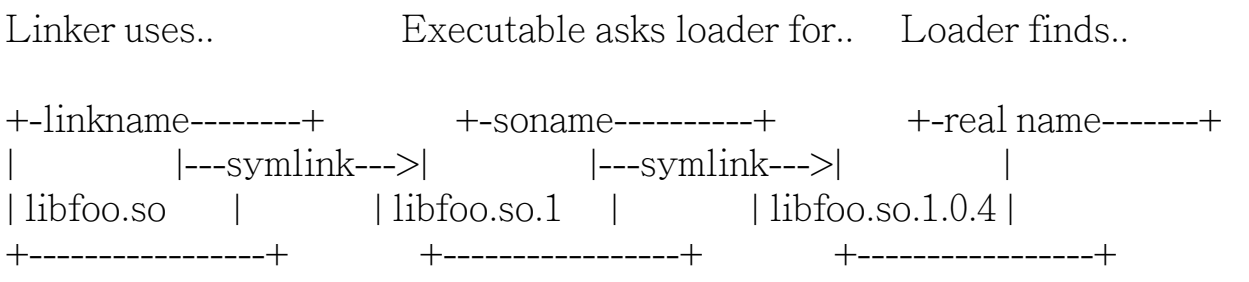
By convention, a shared library has a *major* version number, a *minor* version number and a *patch level* identification (often also a number). The *patch level* is typically changed to specify that a bug fix or minor optimization of the code occurred: version 1.0.12 probably signifies the 12th bug-fix in version 1.0. *Minor* version numbers are typically incremented when a large number of patches have been submitted or a major bug was fixed. *Major* version numbers are changed when the interface to the functions in the shared library changes. Major versions of shared libraries should be backward compatible, hence programs that ran fine using a shared object version 1.0 should also run fine with version 1.1 or with version 1.9.9.

Naming schemes for shared objects

On a Linux system (and on many other Unix systems too) a shared object has a *real name*, a *soname* and a *linkname*. The linkname just contains the name of the library, the soname contains the major version number, and the real name contains the minor version and (when available) the patch level. On a Linux system, the soname and linkname are symbolic links to the correct shared library.

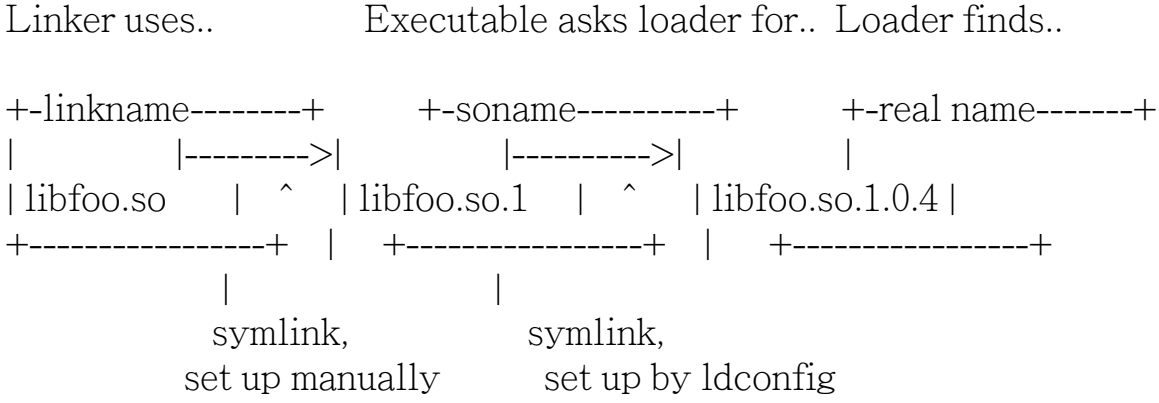
- the *real name* is the name of the file that contains the shared objects precompiled code. By convention the format libNAME.so[.major][.minor][.patchlevel] is used: /lib/libpam.so.0.72; the dynamic loader needs to know this name to find the exact instance of the shared library;
- the *soname* often is the real name without the minor version number and the patch level number: /lib/libpam.so.0. Programs that need a shared object will commonly use the soname to specify the library. On a Linux system that soname is a symbolic link to the real name of the shared library, hence the loader is able to find it;
- the *linkname* is the soname without any versioning information: /lib/libpam.so. The linkname of a shared object is needed as a reference to the shared object during the linking phase of a program (compile time). On a Linux system it is often a link to the soname (which in turn is a link to the real name).

The ASCII art below sketches a typical naming scheme for a shared object on a Linux system in this case for the "foo" library (i.e., version 1.0.4):

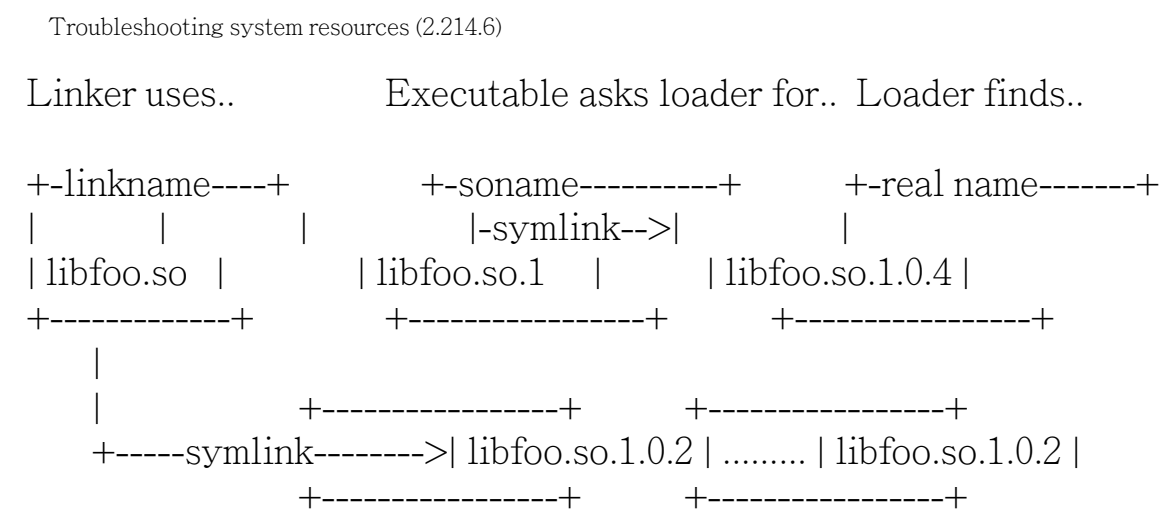


In this case, the compiler/linker and the executable (by means of the run-time loader) will use the exact same shared object, which indeed is the normal situation.

The symbolic link for the soname (which points to the real name) is created automatically when you run the **ldconfig** command (as "root"), which will be discussed in more detail in a later section. The **linkname** is typically set up during installation of a shared library and normally is a symlink to the soname or real name.

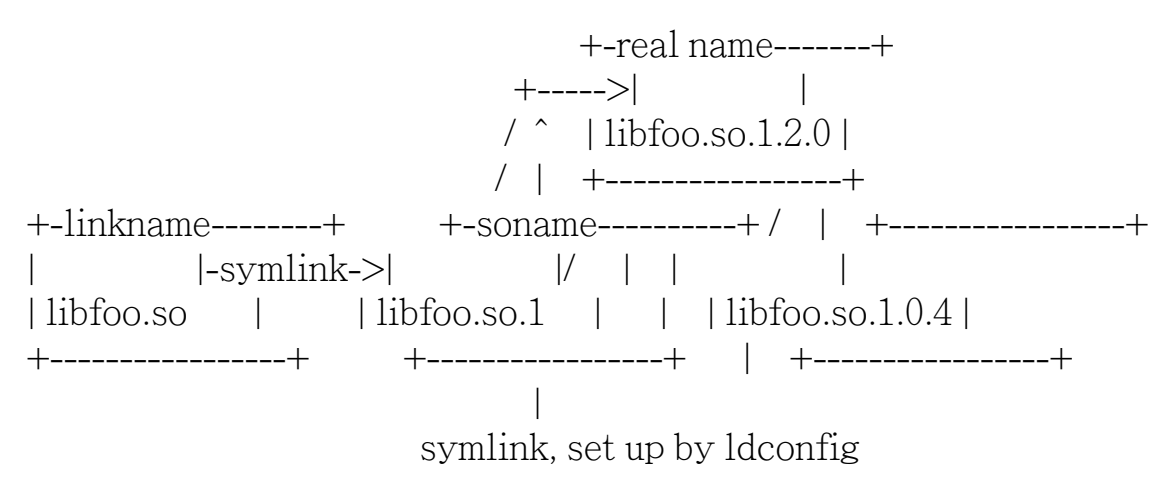


By pointing the symlink to another library, you can develop code using another library:



Now, you can *run* code using one version of a library (the soname symlink will be followed) and develop using another.

Another possibility is upgrading your shared object with a newer one. This typically is done like this:



So: /usr/lib/libfoo.so.2 is a (fully-qualified) *soname*, normally set by **ldconfig** to be a symbolic link to a real name like /usr/lib/libfoo.so.2.0. The linkname would be /usr/lib/libfoo.so, which typically is a symbolic link referring to /usr/lib/libfoo.so.2 too, but may be another name.

ldconfig

The **ldconfig** command performs two important functions: it creates a cache file, that speeds up loading of shared objects and creates the proper symbolic links to enable the linking loader to find the necessary shared object. For a better understanding of how **ldconfig** achieves this, you need to know a bit more about the file format(s) of shared objects.

A shared object is recognizable by its file format. On Intel/Linux that typically is an ELF file format, though the conventional a.out format is still supported on some systems. The

conventional a.out format will not be discussed here in much detail, since on most Unix/Linux systems the ELF file format is standard nowadays.

There are two main stream types of ELF libraries: those linked against libc5 and those linked against libc6 (glibc). Also, there can be shared objects that need *both*.

An ELF file contains a header and various sections; the information contained in the sections depends on the type of ELF file. A shared object however contains a "Dynamic Section" (DS), which in turn contains a number of symbols and corresponding values. One of these symbols is named SONAME and its value contains the soname as specified by the programmer. **ldconfig** uses this value to determine the SONAME for a file that it has detected to be a valid shared object. Also, the files filename is considered.

Note, that the SONAME in the DS is not required to match the (an) actual filename for that shared object e.g., the shared object could be in a file named meaninglessfilename while the SONAME in the DS could be libfoo.so.2.

If you want to inspect the SONAME in the DS, you can use the **objdump** utility:

```
$ objdump -p /usr/lib/libesd.so.0.2.16 |grep SONAME
  SONAME      libesd.so.0
$_
```

ldconfig uses both the SONAME as stored in the shared object *and* the filename it has to create the cache file and the appropriate symbolic links. **ldconfig** scans all directories whose names are passed to it on the command line and in the file /etc/ld.so.conf. The two trusted directories /usr/lib and /lib are always scanned too.

For each directory specified as location for shared objects **ldconfig** scans the files found there. It skips those files it does not recognize as a shared object. Shared objects are recognized as such if they are either in a.out format or in ELF format.

For each shared object its SONAME will be determined next. An internal list is build that contains all SONAMES and the corresponding filename. "Most actual" is determined by the shared library's filename, adhering to the conventions stated before, hence libfoo.so.1.0 is regarded as more recent than libfoo.so.1.

If all shared objects for a given directory have been scanned the appropriate symbolic links will be made: for each SONAME in the list a symbolic link to the most recent version of the shared object (according to its version number) will be created. Next, the cache file will be written, consisting of the SONAMES, type of library and accompanying full file name.

How the dynamic linker locates shared objects

As you know, major versions of shared libraries are backward compatible. Therefore, most programs suffice by specifying the major version number to the dynamic loader (the *soname*).

- the linker for ELF format files, ld-linux.so, starts by checking the environment variable LD_LIBRARY_PATH. This variable may contain a colon separated list of directories to search. Often used to permit non-'root' users to define the location for self-written shared objects, since a non-'root' user is not permitted to put these in the standard locations, or to allow testing;
- if the library could not be found in the directories that are mentioned in the environment variable, the *cache file* is searched. That file, by default /etc/ld.so.cache, contains a compiled list of libraries and typically is created using **ldconfig** as we described earlier. The file /etc/ld.so.cache is not human readable. If you want to see the contents of the cache file use the command **ldconfig -p** /etc/ld.so.cache:

  ```
  # ldconfig -p /etc/ld.so.cache

  432 libs found in cache '/etc/ld.so.cache'
        libzvt.so.2 (libc6) => /usr/lib/libzvt.so.2
        libz.so.1 (libc6) => /usr/lib/libz.so.1
  ...deleted a lot of lines...
        ld-linux.so.2 (ELF) => /lib/ld-linux.so.2
        ld-linux.so.1 (ELF) => /lib/ld-linux.so.1
  ```

- Shared libraries were traditionally stored in /lib and /usr/lib. These locations are searched as a last resort. This will probably never happen unless the cache file does not exist because **ldconfig** has already scanned those directories when creating the cache file.

To see which versions of which shared objects are needed by a program, use the command **ldd**. This command also tries to resolve the matching filename, it uses the cache file to do this. For example:

```
# ldd /usr/bin/vim
      libncurses.so.5 => /lib/libncurses.so.5 (0x4001a000)
      libc.so.6 => /lib/libc.so.6 (0x40063000)
      /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
# _
```

As you see, the author of **vim** choose to use the **ncurses** shared library - to be precise: the library with the *SONAME* libncurses.so.5, which translates to the fully qualified filename /lib/libncurses.so.5.

There are a number of occasions in which the exact location of shared libraries needs to be known. When a program is *linked*, the linker (**ld**) needs their location to be able to refer to them. It searches for them in /lib, /usr/lib and in directories added with -rpath and -Ldir

command line options.

```
gcc -shared -Wl,-soname,libjvc.so.1 -o libjvc.so.1.0.1 jvc.o
ln -sf libjvc.so.1.0.1 libjvc.so.1.0
ln -sf libjvc.so.1.0 libjvc.so.1
ln -sf libjvc.so.1 libjvc.so
```

Prev | Up | Next

General troubleshooting (2.214.5) | Home | Troubleshooting network issues (2.214.7)

Troubleshooting network issues (2.214.7)

Revision: $Revision: 1.6 $

A candidate should be able to identify and correct common network setup issues to include knowledge of locations for basic configuration files and commands.

Key files, terms and utilities include:

/sbin/ifconfig
/sbin/route
/bin/netstat
/etc/network or /etc/sysconfig/network-scripts/
system log files such as /var/log/syslog and /var/log/messages
/bin/ping
/etc/resolv.conf
/etc/hosts
/etc/hosts.allow, /etc/hosts.deny
/etc/hostname or /etc/HOSTNAME
/sbin/hostname
/usr/sbin/traceroute
/usr/bin/nslookup
/usr/bin/dig
/bin/dmesg
host

Something on network troubleshooting in general

The purpose of this chapter is not providing solutions to all possible network problems but more to point out that you have to do it methodically and with knowledge of the surrounding environment.

Knowledge of the surrounding environment is an absolute necessity because you can't solve a problem if you have no idea whatsoever where problems can occur. You have to know how you are connected to the network because every component that plays a role in facilitating your network access can fail.

The key files, terms and utilities will not be described in this chapter since most of them have already been described in Chapter 5, *Networking (2.205)* and other chapters.

An example situation

Let's assume you're working on a PC which is connected to the Internet by means of a local area network and a firewall. Suddenly you can't view that web page anymore.

The first step in determining the problem is to make a list of the components that are involved, this could be something like:

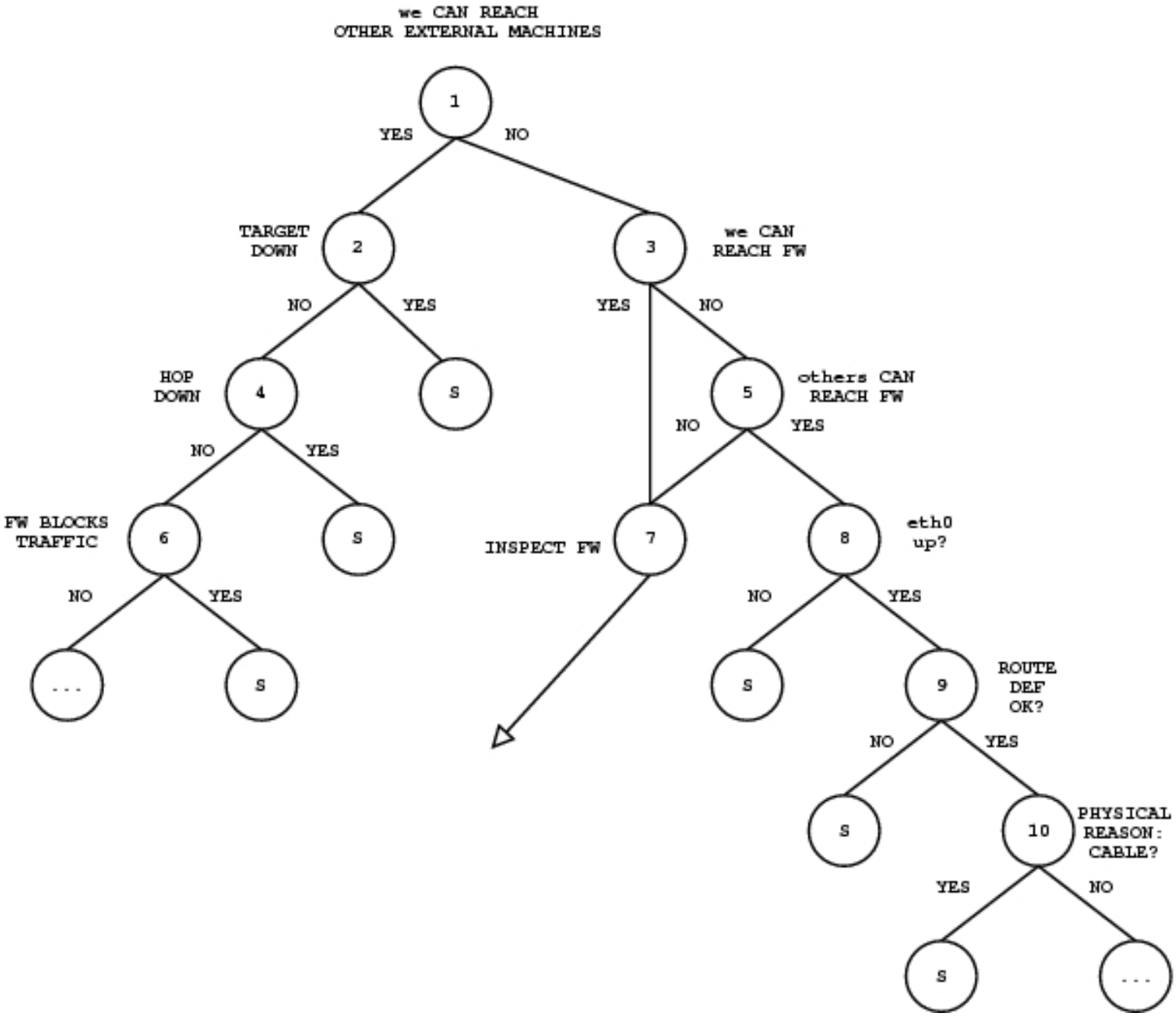
+ your PC with eth0 network interface to the LAN
+ the firewall with eth0 interface to the LAN
  and eth1 interface to the ISP
+ the site you are trying to reach

Then think about how everything works together. You enter the URL in the browser. Your machine uses DNS to find out what the IP address is of the web site you are trying to reach etc.

Packets travel through your eth0 interface over the LAN to the eth0 interface of the firewall and out the eth1 interface of the firewall to the ISP and from the ISP in some way to the web server.

Now that you know how the different components interact, you can take steps to determine the source of the malfunction.

The graphic below should not be treated as *the* way to solve all problems. The purpose of the graphic is to give an example of step-by-step troubleshooting given a certain situation which in this case is fictitious :)



S

The cause of the problem has been determined and can be S(olved).

1

Can we reach other machines on the internet ? Try another URL or try pinging another machine on the internet. Be careful with **ping** though, your firewall could be blocking ICMP echo-requests and replies.

2

Is the machine we are trying to reach, the target, down ? Try reaching the machine via another network, contact a friend and let him try to reach the machine, call the person responsible for the machine etc.

3

Can we reach the firewall? Try pinging the firewall, login to it etc.

4

Is there a HOP down ? Use **traceroute** to find out what the hops are between you and the target host. The route from my machine to LPI's web-server for instance can be determined by issuing the command **traceroute -I www.lpi.org**:

```
# traceroute -I www.lpi.org
traceroute to www.lpi.org (209.167.177.93), 30 hops max, 38 byte packets
 1  fertuut.snowgroningen (192.168.2.1)  0.555 ms  0.480 ms  0.387 ms
 2  wc-1.r-195-85-156.essentkabel.com (195.85.156.1)  30.910 ms  26.352 ms  19.406 ms
 3  HgvL-WebConHgv.castel.nl (195.85.153.145)  19.296 ms  28.656 ms  29.204 ms
 4  S-AMS-IxHgv.castel.nl (195.85.155.2)  172.813 ms  199.017 ms  95.894 ms
 5  f04-08.ams-icr-03.carrier1.net (212.4.194.13)  118.879 ms  84.262 ms  130.855 ms
 6  g02-00.amd-bbr-01.carrier1.net (212.4.211.197)  30.790 ms  45.073 ms  28.631 ms
 7  p08-00.lon-bbr-02.carrier1.net (212.4.193.165)  178.978 ms  211.696 ms  301.321 ms
 8  p13-02.nyc-bbr-01.carrier1.net (212.4.200.89)  189.606 ms  413.708 ms  194.794 ms
 9  g01-00.nyc-pni-02.carrier1.net (212.4.193.198)  134.624 ms  182.647 ms  411.876 ms
10  500.POS2-1.GW14.NYC4.ALTER.NET (157.130.94.249)  199.503 ms  139.083 ms  158.804 ms
11  578.ATM3-0.XR2.NYC4.ALTER.NET (152.63.26.242)  122.309 ms  191.783 ms  297.066 ms
12  188.at-1-0-0.XR2.NYC8.ALTER.NET (152.63.18.90)  212.805 ms  193.841 ms  94.278 ms
13  0.so-2-2-0.XL2.NYC8.ALTER.NET (152.63.19.33)  131.535 ms  131.768 ms  152.717 ms
14  0.so-2-0-0.TL2.NYC8.ALTER.NET (152.63.0.185)  198.645 ms  136.199 ms  274.059 ms
15  0.so-3-0-0.TL2.TOR2.ALTER.NET (152.63.2.86)  232.886 ms  188.511 ms  166.256 ms
16  POS1-0.XR2.TOR2.ALTER.NET (152.63.2.78)  153.015 ms  157.076 ms  150.759 ms
17  POS7-0.GW4.TOR2.ALTER.NET (152.63.131.141)  143.956 ms  146.313 ms  141.405 ms
18  akainn-gw.customer.alter.net (209.167.167.118)  384.687 ms  310.406 ms  302.744 ms
19  new.lpi.org (209.167.177.93)  348.981 ms  356.486 ms  328.069 ms
```

5

Can other machines in the network reach the firewall? Use ping, or login to the firewall from that machine or try viewing a web page on the internet from that machine.

6

Does the firewall block the traffic to that particular machine? Maybe someone doesn't want you to look at that particular site and has instructed the firewall to block traffic to and/or from that site.

7

Inspect the firewall. The problem seems to be on the firewall. Test the interfaces on the firewall, inspect the firewalling rules, check the cabling etc.

8

9 Is our eth0 interface up? This can be tested by issuing the command **ifconfig eth0**.

10 Are your route definitions as they should be? Think of things like default gateway. The route table can be viewed by issuing the command **route -n**.

Is there a physical reason for the problem? Check if the the problem is in the cabling. This could be a defective cable or a badly shielded one. Putting power supply cabling and data cabling through the same tube without metal shielding between the two of them can cause unpredictable, hard to reproduce, errors in the data transmission.

Prev | Up | Next

Troubleshooting system resources (2.214.6) | Home | Troubleshooting environment configurations (2.214.8)

Troubleshooting environment configurations (2.214.8)

Revision: $Revision: 1.5 $

A candidate should be able to identify common local system and user environment configuration issues and common repair techniques.

Key files, terms and utilities include:

/etc/inittab
/sbin/init
/etc/passwd
/etc/shadow
/etc/group
/etc/profile
/etc/rc.local or /etc/rc.boot
/usr/sbin/cron
/usr/bin/crontab
/var/spool/cron/crontabs/
/etc/‘shell_name‘.conf
/etc/login.defs
/etc/syslog.conf

Troubleshooting /etc/inittab and /sbin/init

The program **/sbin/init** reads the file /etc/inittab. See the section called “ What happens next, what does /sbin/init do? ” for a detailed description of the boot process.

If a process isn't running after booting the system, check this file for errors. Then find out what the default runlevel is and check the the /etc/rc?.d/ directory for start/stop scripts.

To get an idea what kind of problems can occur if something is not configured correctly in /etc/inittab, have a look at the file and you'll see things like:

- The default runlevel. This is the runlevel the system will be in when the boot stage is

completed.
- Scripts to run at boottime. These are the scripts in the directory /etc/rcS.d.
- What to do if CTRL+ALT+DEL is pressed. So, if you don't want to have the system respond with a reboot to this famous key combination, you can change this behavior here.
- The number of terminals (Alt+F1...Alt+F*n*) that can be invoked by pressing one of the before mentioned key combinations.
- Getty's on modem lines. A well known message is one of the form: "Id ttyS3 respawning too fast: disabled for 5 minutes".

The *Modem-HOWTO on www.linuxdoc.org* describes this as follows:

> The line mentioned, in this case ttyS3, causes the problem. Make sure the syntax for this line is correct and that the device (ttyS3) exists and can be found. If the modem has negated CD and **getty** opens the port, you'll get this error message since negated CD will kill **getty**. Then **getty** will **respawn** only to be killed again, etc. Thus it respawns over and over (too fast). It seems that if the cable to the modem is disconnected or you have the wrong serial port, it's just like CD is negated. All this can occur when your modem is chatting with **getty**. Make sure your modem is configured correctly. Look at **AT** commands **E** and **Q**.

Troubleshooting authorisation problems

The files involved are /etc/passwd, /etc/shadow and /etc/group.

If a user can't login or gets the wrong shell, chances are the problem is located in /etc/passwd or /etc/shadow. These kind of problems often occurs if someone has been editing the files manually instead of using one of the system tools such as **adduser**, **useradd**, **deluser**, **userdel**.

Suppose, for example, that a user gets the following result when issuing the command **ls -l**:

```
home# ls -l
total 8
dr-xr-xr-x   6 root     0          1024 Dec  6 19:07 ftp
drwxr-xr-x   6 www-data 33         1024 Oct  8 17:18 omproject
drwxr-xr-x   5 piet     1003       1024 Dec  6 23:34 piet
drwxr-xr-x   3 rc564    1001       1024 Dec  4 19:37 rc564
drwxr-xr-x   2 secofr   400        1024 Dec 17 15:26 secofr
drwxr-sr-x  38 willem   1000       3072 Feb  5 09:48 willem
```

Obviously, something has gone wrong: there are group numbers in the result instead of group names as it should be:

```
dr-xr-xr-x    6 root     root        1024 Dec  6 19:07 ftp
drwxr-xr-x    6 www-data www-data    1024 Oct  8 17:18 omproject
drwxr-xr-x    5 piet     piet        1024 Dec  6 23:34 piet
drwxr-xr-x    3 rc564    rc564       1024 Dec  4 19:37 rc564
drwxr-xr-x    2 secofr   secofr      1024 Dec 17 15:26 secofr
drwxr-sr-x   38 willem   willem      3072 Feb  5 09:48 willem
```

The problem lies in /etc/group which is used to replace the group number by the group name. The group is probably not defined in /etc/group.

Troubleshooting /etc/profile

Here you'll often find the system-wide variable definitions such as PATH, **umask** and the prompt. Typical problems that can be the result of an error in /etc/profile can be commands that can't be found (due to a wrong PATH) and new files that are created with the wrong permissions (due to an erroneous **umask**).

Troubleshooting /etc/rc.local or /etc/rc.boot

This is not used anymore as far as I know. Entries that were in these files are know scripts in /etc/rcS.d which should symbolically link to scripts in /etc/init.d as is the case with all the other runlevels.

Troubleshooting cron processes

Cron is responsible for the running of scheduled processes. Each individual user can have his own crontab. If a user can't get his scheduled job to run, the most common causes for this are that the user lacks the authorization to run the command in question or the user lacks the authorization to use cron. The latter is the case if a file /etc/cron.allow exists and the user is not mentioned in that file or if the file /etc/cron.deny exists and the user *is* mentioned in that file.

The files /var/spool/crontabs/<username> which are the user specific cron files, should not be edited directly, use **crontab -e** instead.

Troubleshooting /etc/‘shell_name‘.conf

This has already been described in the section called “Login shells”.

Troubleshooting /etc/login.defs

This has already been described in the section called "Shell startup environment".

Troubleshooting /etc/syslog.conf

This had already been described in the section called "System Logging (2.211.1)".

Chapter 15.

Prev Next

Troubleshooting network issues (2.214.7)

Revision: $Revision: 1.6 $

A candidate should be able to identify and correct common network setup issues to include knowledge of locations for basic configuration files and commands.

Key files, terms and utilities include:

/sbin/ifconfig
/sbin/route
/bin/netstat
/etc/network or /etc/sysconfig/network-scripts/
system log files such as /var/log/syslog and /var/log/messages
/bin/ping
/etc/resolv.conf
/etc/hosts
/etc/hosts.allow, /etc/hosts.deny
/etc/hostname or /etc/HOSTNAME
/sbin/hostname
/usr/sbin/traceroute
/usr/bin/nslookup
/usr/bin/dig
/bin/dmesg
host

Something on network troubleshooting in general

The purpose of this chapter is not providing solutions to all possible network problems but more to point out that you have to do it methodically and with knowledge of the surrounding environment.

Knowledge of the surrounding environment is an absolute necessity because you can't solve a problem if you have no idea whatsoever where problems can occur. You have to know how you are connected to the network because every component that plays a role in facilitating your network access can fail.

The key files, terms and utilities will not be described in this chapter since most of them have already been described in Chapter 5, *Networking (2.205)* and other chapters.

An example situation

Let's assume you're working on a PC which is connected to the Internet by means of a local area network and a firewall. Suddenly you can't view that web page anymore.

The first step in determining the problem is to make a list of the components that are involved, this could be something like:

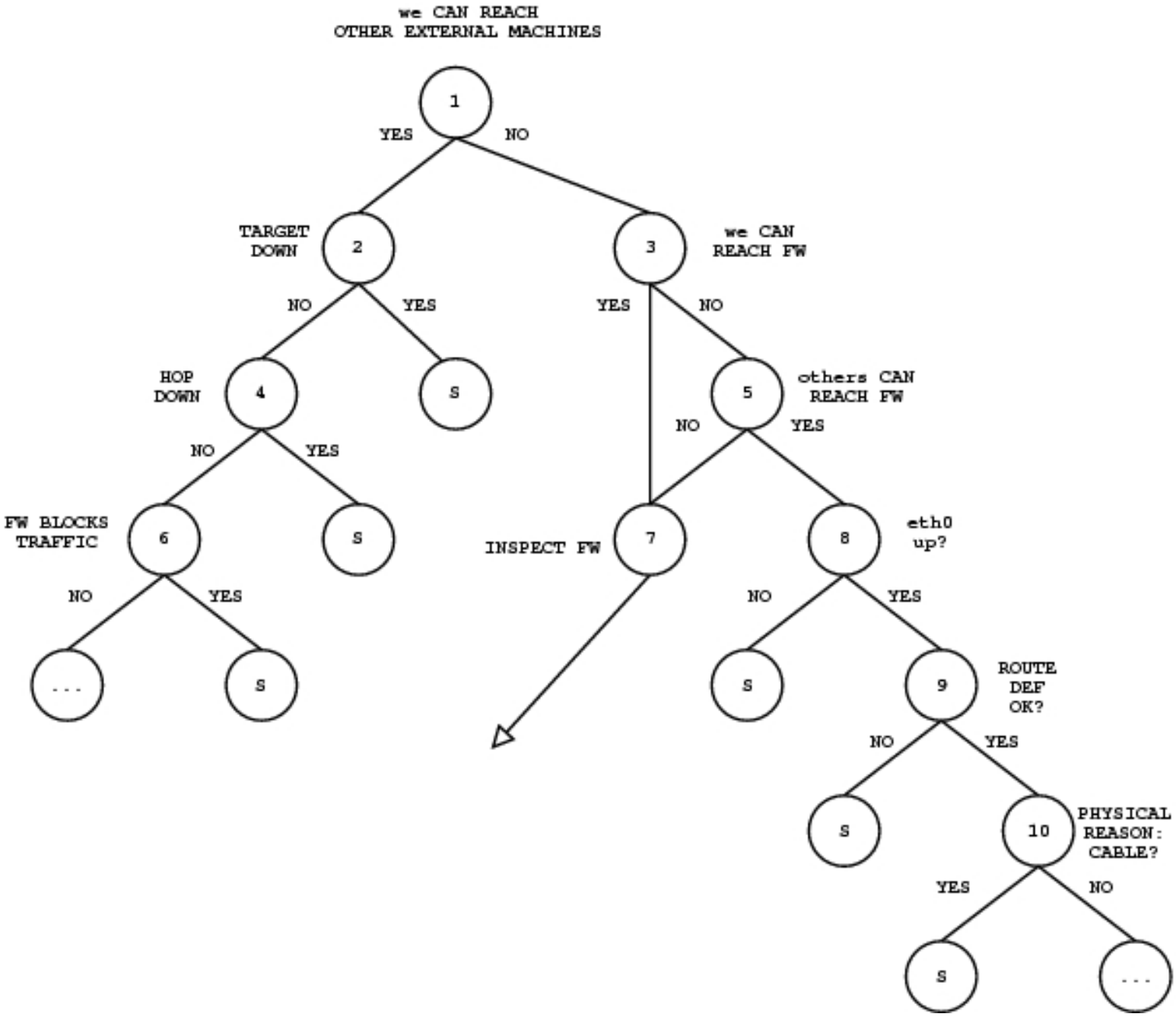
+ your PC with eth0 network interface to the LAN
+ the firewall with eth0 interface to the LAN
  and eth1 interface to the ISP
+ the site you are trying to reach

Then think about how everything works together. You enter the URL in the browser. Your machine uses DNS to find out what the IP address is of the web site you are trying to reach etc.

Packets travel through your eth0 interface over the LAN to the eth0 interface of the firewall and out the eth1 interface of the firewall to the ISP and from the ISP in some way to the web server.

Now that you know how the different components interact, you can take steps to determine the source of the malfunction.

The graphic below should not be treated as *the* way to solve all problems. The purpose of the graphic is to give an example of step-by-step troubleshooting given a certain situation which in this case is fictitious :)



S

The cause of the problem has been determined and can be S(olved).

1

Can we reach other machines on the internet ? Try another URL or try pinging another machine on the internet. Be careful with **ping** though, your firewall could be blocking ICMP echo-requests and replies.

2

Is the machine we are trying to reach, the target, down ? Try reaching the machine via another network, contact a friend and let him try to reach the machine, call the person responsible for the machine etc.

3

Can we reach the firewall? Try pinging the firewall, login to it etc.

4

Is there a HOP down ? Use **traceroute** to find out what the hops are between you and the target host. The route from my machine to LPI's web-server for instance can be determined by issuing the command **traceroute -I www.lpi.org**:

```
# traceroute -I www.lpi.org
traceroute to www.lpi.org (209.167.177.93), 30 hops max, 38 byte packets
 1  fertuut.snowgroningen (192.168.2.1)  0.555 ms  0.480 ms  0.387 ms
 2  wc-1.r-195-85-156.essentkabel.com (195.85.156.1)  30.910 ms  26.352 ms  19.406 ms
 3  HgvL-WebConHgv.castel.nl (195.85.153.145)  19.296 ms  28.656 ms  29.204 ms
 4  S-AMS-IxHgv.castel.nl (195.85.155.2)  172.813 ms  199.017 ms  95.894 ms
 5  f04-08.ams-icr-03.carrier1.net (212.4.194.13)  118.879 ms  84.262 ms  130.855 ms
 6  g02-00.amd-bbr-01.carrier1.net (212.4.211.197)  30.790 ms  45.073 ms  28.631 ms
 7  p08-00.lon-bbr-02.carrier1.net (212.4.193.165)  178.978 ms  211.696 ms  301.321 ms
 8  p13-02.nyc-bbr-01.carrier1.net (212.4.200.89)  189.606 ms  413.708 ms  194.794 ms
 9  g01-00.nyc-pni-02.carrier1.net (212.4.193.198)  134.624 ms  182.647 ms  411.876 ms
10  500.POS2-1.GW14.NYC4.ALTER.NET (157.130.94.249)  199.503 ms  139.083 ms  158.804 ms
11  578.ATM3-0.XR2.NYC4.ALTER.NET (152.63.26.242)  122.309 ms  191.783 ms  297.066 ms
12  188.at-1-0-0.XR2.NYC8.ALTER.NET (152.63.18.90)  212.805 ms  193.841 ms  94.278 ms
13  0.so-2-2-0.XL2.NYC8.ALTER.NET (152.63.19.33)  131.535 ms  131.768 ms  152.717 ms
14  0.so-2-0-0.TL2.NYC8.ALTER.NET (152.63.0.185)  198.645 ms  136.199 ms  274.059 ms
15  0.so-3-0-0.TL2.TOR2.ALTER.NET (152.63.2.86)  232.886 ms  188.511 ms  166.256 ms
16  POS1-0.XR2.TOR2.ALTER.NET (152.63.2.78)  153.015 ms  157.076 ms  150.759 ms
17  POS7-0.GW4.TOR2.ALTER.NET (152.63.131.141)  143.956 ms  146.313 ms  141.405 ms
18  akainn-gw.customer.alter.net (209.167.167.118)  384.687 ms  310.406 ms  302.744 ms
19  new.lpi.org (209.167.177.93)  348.981 ms  356.486 ms  328.069 ms
```

5

Can other machines in the network reach the firewall? Use ping, or login to the firewall from that machine or try viewing a web page on the internet from that machine.

6

Does the firewall block the traffic to that particular machine? Maybe someone doesn't want you to look at that particular site and has instructed the firewall to block traffic to and/or from that site.

7

Inspect the firewall. The problem seems to be on the firewall. Test the interfaces on the firewall, inspect the firewalling rules, check the cabling etc.

8

9 Is our eth0 interface up? This can be tested by issuing the command **ifconfig eth0**.

10 Are your route definitions as they should be? Think of things like default gateway. The route table can be viewed by issuing the command **route -n**.

Is there a physical reason for the problem? Check if the the problem is in the cabling. This could be a defective cable or a badly shielded one. Putting power supply cabling and data cabling through the same tube without metal shielding between the two of them can cause unpredictable, hard to reproduce, errors in the data transmission.

Chapter 16.

Prev Next

Troubleshooting environment configurations (2.214.8)

Revision: $Revision: 1.5 $

A candidate should be able to identify common local system and user environment configuration issues and common repair techniques.

Key files, terms and utilities include:

/etc/inittab
/sbin/init
/etc/passwd
/etc/shadow
/etc/group
/etc/profile
/etc/rc.local or /etc/rc.boot
/usr/sbin/cron
/usr/bin/crontab
/var/spool/cron/crontabs/
/etc/‘shell_name‘.conf
/etc/login.defs
/etc/syslog.conf

Troubleshooting /etc/inittab and /sbin/init

The program **/sbin/init** reads the file /etc/inittab. See the section called “ What happens next, what does /sbin/init do? ” for a detailed description of the boot process.

If a process isn't running after booting the system, check this file for errors. Then find out what the default runlevel is and check the the /etc/rc?.d/ directory for start/stop scripts.

To get an idea what kind of problems can occur if something is not configured correctly in /etc/inittab, have a look at the file and you'll see things like:

- The default runlevel. This is the runlevel the system will be in when the boot stage is

completed.
- Scripts to run at boottime. These are the scripts in the directory /etc/rcS.d.
- What to do if CTRL+ALT+DEL is pressed. So, if you don't want to have the system respond with a reboot to this famous key combination, you can change this behavior here.
- The number of terminals (Alt+F1...Alt+F*n*) that can be invoked by pressing one of the before mentioned key combinations.
- Getty's on modem lines. A well known message is one of the form: "Id ttyS3 respawning too fast: disabled for 5 minutes".

The *Modem-HOWTO on www.linuxdoc.org* describes this as follows:

> The line mentioned, in this case ttyS3, causes the problem. Make sure the syntax for this line is correct and that the device (ttyS3) exists and can be found. If the modem has negated CD and **getty** opens the port, you'll get this error message since negated CD will kill **getty**. Then **getty** will **respawn** only to be killed again, etc. Thus it respawns over and over (too fast). It seems that if the cable to the modem is disconnected or you have the wrong serial port, it's just like CD is negated. All this can occur when your modem is chatting with **getty**. Make sure your modem is configured correctly. Look at **AT** commands **E** and **Q**.

Troubleshooting authorisation problems

The files involved are /etc/passwd, /etc/shadow and /etc/group.

If a user can't login or gets the wrong shell, chances are the problem is located in /etc/passwd or /etc/shadow. These kind of problems often occurs if someone has been editing the files manually instead of using one of the system tools such as **adduser**, **useradd**, **deluser**, **userdel**.

Suppose, for example, that a user gets the following result when issuing the command **ls -l**:

```
home# ls -l
total 8
dr-xr-xr-x   6 root     0          1024 Dec  6 19:07 ftp
drwxr-xr-x   6 www-data 33         1024 Oct  8 17:18 omproject
drwxr-xr-x   5 piet     1003       1024 Dec  6 23:34 piet
drwxr-xr-x   3 rc564    1001       1024 Dec  4 19:37 rc564
drwxr-xr-x   2 secofr   400        1024 Dec 17 15:26 secofr
drwxr-sr-x  38 willem   1000       3072 Feb  5 09:48 willem
```

Obviously, something has gone wrong: there are group numbers in the result instead of group names as it should be:

```
dr-xr-xr-x    6 root     root        1024 Dec  6 19:07 ftp
drwxr-xr-x    6 www-data www-data    1024 Oct  8 17:18 omproject
drwxr-xr-x    5 piet     piet        1024 Dec  6 23:34 piet
drwxr-xr-x    3 rc564    rc564       1024 Dec  4 19:37 rc564
drwxr-xr-x    2 secofr   secofr      1024 Dec 17 15:26 secofr
drwxr-sr-x   38 willem   willem      3072 Feb  5 09:48 willem
```

The problem lies in /etc/group which is used to replace the group number by the group name. The group is probably not defined in /etc/group.

Troubleshooting /etc/profile

Here you'll often find the system-wide variable definitions such as PATH, **umask** and the prompt. Typical problems that can be the result of an error in /etc/profile can be commands that can't be found (due to a wrong PATH) and new files that are created with the wrong permissions (due to an erroneous **umask**).

Troubleshooting /etc/rc.local **or** /etc/rc.boot

This is not used anymore as far as I know. Entries that were in these files are know scripts in /etc/rcS.d which should symbolically link to scripts in /etc/init.d as is the case with all the other runlevels.

Troubleshooting cron processes

Cron is responsible for the running of scheduled processes. Each individual user can have his own crontab. If a user can't get his scheduled job to run, the most common causes for this are that the user lacks the authorization to run the command in question or the user lacks the authorization to use cron. The latter is the case if a file /etc/cron.allow exists and the user is not mentioned in that file or if the file /etc/cron.deny exists and the user *is* mentioned in that file.

The files /var/spool/crontabs/<username> which are the user specific cron files, should not be edited directly, use **crontab -e** instead.

Troubleshooting /etc/'shell_name'.conf

This has already been described in the section called "Login shells".

Troubleshooting /etc/login.defs

This has already been described in the section called "Shell startup environment".

Troubleshooting /etc/syslog.conf

This had already been described in the section called "System Logging (2.211.1)".

Appendix A. LPIC Level 2 Objectives

Prev Next

Appendix A. LPIC Level 2 Objectives

Table A.1. LPIC Level 2.201 - 2.205 Objectives And Their Relative Weight

2	LPIC Level 2	100
201	Linux Kernel	:
2.201.1	Kernel Components	1
2.201.2	Compiling a kernel	1
2.201.3	Patching a kernel	2
2.201.4	Customising a kernel	1
202	System Startup	:
2.202.1	Customising system startup and boot processes	2
2.202.2	System recovery	3
203	Filesystem	:
2.203.1	Operating the Linux filesystem	3
2.203.2	Maintaining a Linux filesystem	4
2.203.3	Creating and configuring filesystem options	3
204	Hardware	:
2.204.1	Configuring RAID	2
2.204.2	Adding new hardware	3
2.204.3	Software and kernel configuration	2
2.204.4	Configuring PCMCIA devices	1
205	Networking Configuration	:
2.205.1	Basic networking configuration	5
2.205.2	Advanced Network Configuration and Troubleshooting	3

Table A.2. LPIC Level 2.206 - 2.209 Objectives And Their Relative Weight

206	Mail & News	:

2.206.1	Configuring mailing lists	1
2.206.2	Using email servers	4
2.206.3	Managing Mail Traffic	3
2.206.4	Serving news	1
207	DNS	:
2.207.1	Basic DNS server configuration	2
2.207.2	Create and maintain DNS zones	3
2.207.3	Securing a DNS server	3
208	Web Services	:
2.208.1	Implementing a web server	2
2.208.2	Maintaining a web server	2
2.208.3	Implementing a proxy server	2
209	File and Service Sharing	:
2.209.1	Configuring a Samba server	5
2.209.2	Configuring an NFS server	3

Table A.3. LPIC Level 2.210 - 2.214 Objectives And Their Relative Weight

210	Network Client Management	:
2.210.1	DHCP configuration	2
2.210.2	NIS configuration	1
2.210.3	LDAP configuration	1
2.210.4	PAM authentication	2
211	System Maintenance	:
2.211.1	System logging	1
2.211.2	Packaging software	1
2.211.3	Backup operations	2
212	System Security	:
2.212.2	Configuring a router	2
2.212.3	Securing FTP servers	2
2.212.4	Secure shell (SSH)	2
2.212.5	TCP_wrappers	1

2.212.6	Security tasks	3
213	System Customization and Automation	:
2.213.1	Automating tasks using scripts	3
214	Troubleshooting	:
2.214.2	Creating recovery disks	1
2.214.3	Identifying boot stages	1
2.214.4	Troubleshooting bootloaders	1
2.214.5	General troubleshooting	1
2.214.6	Troubleshooting system resources	1
2.214.7	Troubleshooting network issues	1
2.214.8	Troubleshooting environment configurations	1

Bibliography

Prev Next

Bibliography

[Albitz01] Paul Albitz. Cricket Liu. *DNS and BIND (Fourth Edition)* . O'Reilly. 2001. ISBN 0-596-00158-4.

[Bandel97] David Bandel. *Disk Maintenance under Linux (Disk Recovery)* . Linux Journal. January 1997.

[Bar00] Moshe Bar. *Using RAID with Linux* . Dr Dobbs. March 29th, 2000.

[BINDfaq] *Frequently Asked Questions about BIND* . Nominum.

[BootFAQ] *The Linux Bootdisk HOWTO* .

[Bird01] Tina Bird. *Virtual Private Networks FAQ* . Aug 2001.

[Brockmeier01] Joe Brockmeier. *A Look at Squid* . Unix review.

[Coar00] Ken Coar. *Security and Apache: An Essential Primer*, . INT Media Group, Incorporated. February 2000.

[Colligan00] Tom Colligan. *Ultra ATA-100 Extends Ultra ATA-66 Interface*, . Dell. August 2000.

[Dawson00] Terry Dawson. Olaf Kirch. *The Linux Network Administrator's Guide, Second Edition*, . Linux Documentation Project. March 2000.

[Dean01] Jeffrey Dean. *LPI Linux Certification In A Nutshell, a Desktop Quick Reference*. O'Reilly. June 2001, first edition.

[DNShowto] Nicolai Langfeldt. Jamie Norrish. *DNS HOWTO* . v3.1: October 2001.

[Don99] don@sabotage.org. *Automount mini-Howto* . April 1999.

[Drake00] Joshua Drake. *Linux Networking HOWTO* . December 2000.

[Engelschall00] Ralph S. Engelschall. *User manual for mod_ssl 2.6* . 2000.

[FreeSWAN] *FreeSWAN HTML Documentation* .

[Friedl01] Jeffrey E.F. Friedl. *Mastering Regular Expressions* . O'Reilly. 1997.. ISBN 0-56592-257-3.

[Hinds01] David Hinds. *Linux PCMCIA HOWTO* . Sourceforge.net. July 13, 2001.

[Hubert00] Bert Hubert. Richard Allen. *Logical Volume Manager HOWTO* . Linux Documentation Project. April 28, 2000.

[Impson00] Jeremy Impson. *Linux Power Tuning*, . Network Computing. November 13, 2000.

[Jackson01] Ian Jackson. *Debian Policy Manual* . 24 July 2001.

[Johnson01] Richard B. Johnson. *The Linux-Kernel Archive* . April 1999.

[Kiracofe01] Daniel Kiracofe. *Transparent Proxy with Squid mini-HOWTO* . v1.3, January 2001.

[Krause01] Ralph Krause. *Installing PHP4 and MySQL* . January 10th, 2001.

[Lindgreen01] Ted Lindgreen. *Current status and future of DNSSEQ. Talk presented at the Snow Unix Event, September 14 2001*. . Snow B.V.. 14 september 2001.

[LinuxRef01] *Red Hat Linux 6.2: The official Red Hat Linux Reference Guide*, . Red Hat.

[LinuxRef02] *Red Hat hardware guide: **lspci***, . Red Hat.

[LinuxRef03] *Adding a Hard drive in Linux -- In five steps*, . Linuxnewbie.org.

[LinuxRef04] *Configuring the Serial Hardware - Linux Network Administrators Guide*, . LDP.

[LinuxRef05] *XFree86 Video Timings HOWTO*, . XFree86 Project Inc..

[LinuxRef06] *Apache Server FAQ*, . Apache. February 28, 2001.

[LinuxRef07] *Introduction to Linux Systems Administration*, . Central Queensland University. 1999.

[LinuxRef08] *Apache Virtual Host Documentation*, . Apache.

[Liu00] Cricket Liu. *Securing an Internet Name Server (pdf)* , . Acme Byte and Wire LLC. 2000.

[Lugo00] David Lugo. *Dual chrooted BIND/DNS servers* .

[McGough01] Nancy McGough. *Procmail Quick Start* . Copyright © 1994 Nancy McGough and Infinite Ink.

[NFS] *NFS project at Sourceforge*, . Open Source Developer Network.

[NFSv4] *NFS version 4* .

[Nielsen98] Mark Nielsen. *Quick autofs Tutorial*, . Linux Gazette. Januari 1998.

[Nielsen01] Mark Nielsen. *Installing USB and Upgraded Kernel 2.2.18 for my laptop*, . GNUJobs.com. February 14, 2001.

[Nijssen99] Theo Nijssen. *CERT-NL - Security Advisory: Domain Name System (DNS) Denial of Service* , . CERT NL. August 1999.

[Pearson00] Oskar Pearson. *Squid Users guided*, . Squid-cache.org. September 28, 2000.

[PerlRef01] Stas Bekman. *Mod_perl guide*, . The Apache Group. September 2001.

[PerlRef02] *Perlsec - Perl security*, . ActivePerl Documentation.

[PerlRef03] Neal Holz. *The Perl programming language*, . Civil Engineering at Carleton University, Ottawa, Canada.. Februari 1995.

[PerlRef04] Randal L. Schwartz. Tom Phoenix. *Learning Perl, example chapter, 3rd edition* . O'Reilly. July 2001.

[Poet99] Poet. *Linux WWW HOWTO*, . Linux Review. August 21, 1999.

[Prasad01] Tanmoy Prasad. Chris Snell. *The Linux Alphanumeric Pager Gateway Mini-HOWTO*, . Linux Documentation Project. July 25, 2001.

[Raftery01] James Raftery. *Securing DNS with Transaction Signatures* , . *ILUG* . January 2001.

[Robbins01] Daniel Robbins. *Software RAID in the new Linux 2.4 kernel (part 1)*, . IBM DeveloperWorks. February 2001.

[Robbins01.2] Daniel Robbins. *Linux Hardware Stability guide (part 2)*, . IBM DeveloperWorks. July 2001.

[Robbins96] Arnold D. Robbins. *A user's guide to awk*, . 2001.

[Samsonova00] Elena Samsonova. *INN -- InterNetNews*, . July 6, 2000.

[Sayle98] Robert P. Sayle. *Bourne Shell Programming*, . June 6, 1998.

[Till01] David Till. *Teach Yourself Perl 5 in 21 days, Second Edition*, . SAMS.Net/Macmillan Computer Publishing. 0-672-30894-0.

[Truemper00] Winfried Truemper. *CD-Writing HOWTO*, . July 23, 2000.

[USBRef01] *The Linux USB Project*, . linux-usb.org.

[Vasudevan02] Alavoor Vasudevan. *LILO, Linux Crash Rescue HOW-TO*, . linuxdoc.org. January 2002.

[Wall01] Larry Wall. Tom Christiansen. Jon Orwant. *Programming Perl (3rd edition)*, . O'Reilly. 2000.. ISBN 0-596-00027-8.

[Wessels01] Duane Wessels. *SQUID Frequently Asked Questions* .

[Will00] Michael Will. *LDP: the Linux PCI-HOWTO* . Linux Documentation Project.

[Wilson00] Brian Wilson. *Monitoring Apache With Cricket* . O'Reilly OnLamp.com. March 17, 2000.

[Wirzenius98] Lars Wirzenius. Joanna Oja. Stephen Stafford. *The Linux System Administrator's Guide* . Linux Documentation Project. Version 0.7.

[Yap98] Ken Yap. *An Introduction to Network Booting and Etherboot* . Linux Focus. September 1998.

[Zadok01] Erez Zadok. *Linux NFS and Automounter Administration (Craig Hunt Linux Library)*, . Sybex. 2001.. ISBN 0-7821-2739-8.

[Sharpe01] Richard Sharpe. *Using Samba, Special Edition*. Que Corporation. July 2000, first

printing.

Index

Prev

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

z
