The LPIC-2 Exam Prep

6th edition, for version 4.5

COLLABORATORS

	TITLE : The LPIC-2 Exam Prep	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Written, updated and reviewed by many, many Snow B.V. colleagues. , Jos Jansen, and Joost Helberg	2017	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Figures

List of Tables

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 to 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 acquired from material on the Internet. Also 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 at lpic2-editor@snow.nl so we can take corrective actions.

Organization of this book: This book has been organized to follow the Linux Professional Institute level 2 objectives for LPIC-2 certification revision 4.5.0 of February 13th, 2017. The detailed objectives are available via https://www.lpi.org/study-resources/-lpic-2-201-exam-objectives/ and https://www.lpi.org/study-resources/lpic-2-202-exam-objectives/ .

In case new objectives are published, the book will follow shortly thereafter. The book for version 4 of the objectives is still available but is not maintained anymore and will disappear some time after version 4.5 is active.

The authors use the DocBook documentation standard for this book.

Preface

This book reflects the efforts of a number of experienced Linux professionals to prepare for the LPIC-2 exam. It is – and always will be – a work in progress. The authors previously obtained their LPIC-1 levels, and wanted to participate in the 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 milestones 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.

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 its closed-source alternatives. Driven by closed source vendors' questionable license policies, security risks, bugs and vendor-lock, more and more IT-managers have chosen 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 even if that admin would fall on them. These recruiters need a way to certify and recognize the skills of their candidates. Also the candidates need a way to prove their skills, both to themselves and to the recruiter.

A number of organizations offer Linux certifications. 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 of quality and independency. They also are a part of our community so 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 we started a month before that. 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

Chapter 0

Capacity Planning (200)

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

Objective 200.1; Measure and Troubleshoot Resource Usage (6 points) Candidates should be able to measure hardware resource and network bandwidth, identify and troubleshoot resource problems.

Objective 200.2; Predict Future Resource Needs (2 points) Candidates should be able to monitor resource usage to predict future resource needs.

Measure and Troubleshoot Resource Usage (200.1)

Objectives

Candidates should be able to measure hardware resource and network bandwidth, identify and troubleshoot resource problems.

KEY KNOWLEDGE AREAS:

- Measure CPU Usage
- Measure memory usage
- Measure disk I/O
- Measure network I/O
- Measure firewalling and routing throughput
- Map client bandwith usage.
- Match / correlate system symptoms with likely problems
- Estimate throughput and identify in a system including networking

THE FOLLOWING IS A PARTIAL LIST OF USED FILES, TERMS AND UTILITIES:

- **iostat**
- **iotop**
- **vmstat**
- **netstat**
- **ss**

- **iptraf**
- **pstree, ps**
- **w**
- **lsof**
- **top**
- **htop**
- **uptime**
- **sar**
- swap
- processes blocked on I/O
- blocks in
- blocks out
- network

iostat

Note
Depending on the version and distribution of your Linux version it may be necessary to install a package like Debian `sysstat` to install tools like **iostat**, **sar**, **mpstat** etc. The Debian `procps` package contains utilities like **free**, **uptime**, **vmstat**, **w**, **sysctl** etc.

The **iostat** command is used for monitoring system input/output (I/O) device load. This is done by observing *the time* the devices are active in relation to their *average* transfer rates. The **iostat** command without any options shows the **iostat** statistics since the last system reboot. The **iostat** command with sequence options, using interval and count, first shows the statistics since the last system reboot, unless this is omitted using the `-y` option, followed by the statistics during each specified interval.

Usage:

```
$ iostat [options] [interval [count] ]
```

Examples:

```
$ iostat
Linux 3.2.0-4-686-pae (debian)  05/07/2013  _i686_  (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
1.25    0.32    3.76    0.20    0.00   94.46

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda              12.21       214.81        17.38     333479      26980

$ iostat 1 3
Linux 3.2.0-4-686-pae (debian)  05/07/2013  _i686_  (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
24.24    1.51    7.49    4.97    0.00   61.79

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda              14.06       249.94       121.22    1337998     648912
```

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
13.13    0.00    6.21    0.60    0.00   80.06

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               2.51         4.01        28.86         40        288

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
11.11    0.00    5.71    0.00    0.00   83.18

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.30         0.00         3.20          0         32

$ iostat -c
Linux 3.2.0-4-686-pae (debian)  05/07/2013  _i686_  (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
9.16    0.05   17.01    1.37    0.00   72.42
```

iotop

The **iotop** command is similar to the **top** command. It shows I/O usage information output by the Linux kernel and displays a table of current I/O usage by processes or threads on the system.

iotop displays columns for the I/O bandwidth read and written by each process/thread during the sampling period. It also displays the percentage of time the thread/process spent while swapping in and while waiting on I/O. For each process, its I/O priority (class/level) is shown. In addition, the total I/O bandwidth read and written during the sampling period is displayed at the top of the interface.

Usage:

```
$ iotop [options]
```

Example:

```
  $ iotop -b |head
Total DISK READ :       0.00 B/s | Total DISK WRITE :     213.38 M/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:       0.00 B/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN      IO    COMMAND
    6 be/4 root        0.00 B/s    0.00 B/s  0.00 % 99.99 % [kworker/u8:0]
  191 be/3 root        0.00 B/s    0.00 B/s  0.00 % 94.14 % [jbd2/sda1-8]
 2976 be/4 root        0.00 B/s  213.38 M/s  0.00 %  0.00 % dd if=/dev/zero of=/tmp/foo
    1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % init
    2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
    3 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
    5 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/0:0H]
```

vmstat

The **vmstat** command reports virtual memory statistics about processes, memory, paging, block IO, traps, and CPU utilization.

Usage:

```
$ vmstat [options] [delay [count]]
```

Example:

```
$ vmstat 2 2
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
0  0      0 109112  33824 242204    0    0   603    26  196  516  7 11 81  1
0  0      0 109152  33824 242204    0    0     0     2  124  239  0  1 98  0
```

Please beware that the first row will always show average measurements since the machine has booted, and should therefore be neglected. Values which are related to memory and I/O are expressed in kilobytes (1024 bytes). Old (pre-2.6) kernels might report blocks as 512, 2048 or 4096 bytes instead. Values related to CPU measurements are expressed as a percent of *total* CPU time. Keep this in mind when interpreting measurements from a multi-CPU system. - on these systems **vmstat** averages the number of CPUs into the output. All five CPU fields should add up to a total of 100% for each interval shown. This is independent of the number of processors or cores. In other words: **vmstat** can NOT be used to show statistics per processor or core (**mpstat** or **ps** should be used in that case). **vmstat** will accept delay in seconds and a number of counts (repetitions) as an argument, but the process and memory measurement results will always remain to be instantaneous.

The first process column, "r" lists the number of processes currently allocated to the processor*run* queue. These processes are waiting for processor run time, also known as CPU time.

The second process column, "b" lists the number of processes currently allocated to the *block* queue. These processes are listed as being in *uninterruptable sleep*, which means they are waiting for a device to return either input or output (I/O).

The first memory column, "swpd" lists the amount of virtual memory being used expressed in kilobytes (1024 bytes). Virtual memory consists of swap space from disk, which is considerably slower than physical memory allocated inside memorychips.

The second memory column, "free" lists the amount of memory currently not in use, not cached and not buffered expressed.

The third memory column, "buff" lists the amount of memory currently allocated to buffers. Buffered memory contains raw disk blocks.

The fourth memory column, "cache" lists the amount of memory currently allocated to caching. Cached memory contains files.

The fifth memory column, "inact" lists the amount of inactive memory. This is only shown using the `-a` option.

The sixth memory column, "active" lists the amount of active memory. This is only shown using the `-a` option.

The first swap column, "si" lists the amount of memory being swapped *in* from disk (per second).

The second swap column, "so" lists the amount of memory being swapped *out* to disk (per second).

The first io column, "bi" lists the amount of blocks per second being received from a block device.

The second io column, "bo" lists the amount of blocks per second being sent to a block device.

The first system column, "in" lists the number of interrupts per second (including the clock).

The second system column, "cs" lists the number of context switches per second.

The cpu columns are expressed as percentages of total CPU time.

The first cpu column, "us" (user code) shows the percentage of time spent running non-kernel code.

The second cpu column, "sy" (system code) shows the percentage of time spent running kernel code.

The third cpu column, "id" shows the percentage of idle time.

The fourth cpu column, "wa" shows the percentage of time spent waiting for I/O (Input/Output).

The fifth cpu column, "st" (steal time) shows the percentage of time stolen from a virtual machine. This is the amount of real CPU time the virtual machine (hypervisor or VirtualBox) has allocated to tasks other than running your virtual machine.

netstat

The **netstat** command shows network connections, routing tables, interface statistics, masquerade connections and multicast memberships. The results are dependant on the first argument:

- **(no argument given)** - all active sockets of all configured address families will be listed.

- `--route, -r` - the kernel routing tables are shown, output is identical to **route -e** (note: in order to use **route**, elevated privileges might be needed whereas **netstat -r** can be run with user privileges instead).
- `--groups, -g` - lists multicast group membership information for IPv4 and IPv6
- `--interfaces, -i` - lists all network interfaces and certain specific properties
- `--statistics, -s` - lists a summary of statistics for each protocol, similar to SNMP output
- `--masquerade, -M` - lists masqueraded connections on pre-2.4 kernels. On newer kernels, use **cat /proc/net/ip_conntrack** instead. In order for this to work, the *ipt_MASQUERADE* kernel module has to be loaded. This applies to 2.x and 3.x kernels.

Usage:

```
$ netstat [address_family_options] [options]
```

Examples:

```
$ netstat -aln --tcp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:34236           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:389             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:25                  :::*                    LISTEN
tcp6       0      0 :::32831                :::*                    LISTEN

$ netstat -al --tcp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 localhost:smtp          *:*                     LISTEN
tcp        0      0 *:34236                 *:*                     LISTEN
tcp        0      0 *:ldap                  *:*                     LISTEN
tcp        0      0 *:sunrpc                *:*                     LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 localhost:smtp          [::]:*                  LISTEN
tcp6       0      0 [::]:32831              [::]:*                  LISTEN
```

ss

The **ss** command is used to show socket statistics. It can display stats for PACKET sockets, TCP sockets, UDP sockets, DCCP sockets, RAW sockets, Unix domain sockets, and more. It allows showing information similar to the **netstat** command, but it can display more TCP and state information.

Most Linux distributions are shipped with ss. Being familiar with this tool helps enhance your understand of what's going on in the system sockets and helps you find the possible causes of a performance problem.

Usage:

```
$ ss [options] [filter]
```

Example:

```
$ ss -t -a
State      Recv-Q Send-Q              Local Address:Port             Peer Address:Port
LISTEN     0      5                   192.168.122.1:domain                  *:*
LISTEN     0      128                            *:ssh                       *:*
LISTEN     0      128                     127.0.0.1:ipp                      *:*
LISTEN     0      100                     127.0.0.1:smtp                     *:*
```

iptraf

iptraf is a network monitoring utility for IP networks and can be used to monitor the load on an IP network. It intercepts packets on the network and gives out various pieces of information about the current IP traffic over it.

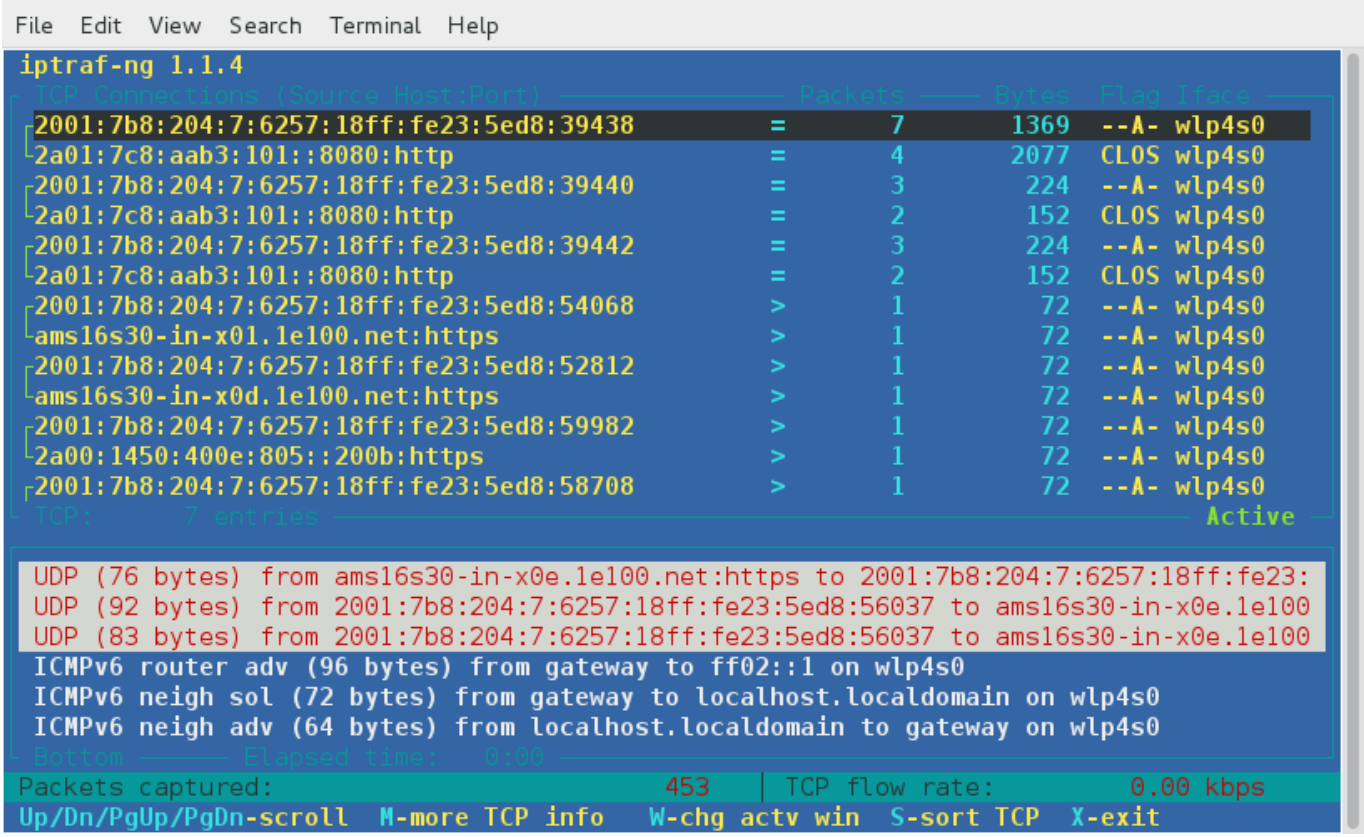
iptraf gathers data like TCP connection packet and byte counts, interface statistics and activity indicators, TCP/UDP traffic breakdowns, and LAN station packet and byte counts. IPTraf features include an IP traffic monitor which shows TCP flag information, packet and byte counts, ICMP details, OSPF packet types, and oversized IP packet warnings.

Usage:

```
$ iptraf [options]
```

Example:

```
$ iptraf
```



*The **iptraf** window.*

ps

Usage:

```
$ ps [options]
```

The **ps** command shows a list of the processes currently running. These are the same processes which are being shown by the **top** command. The GNU version of **ps** accepts three different *kind* of options:

1. UNIX options - these may be grouped and *must* be preceded by a dash
2. BSD options - these may be grouped and must be used *without* a dash

3. GNU long options - these are preceded by *two* dashes

These options may be mixed on GNU **ps** up to some extent, but bare in mind that depending on the version of Linux you are working on you might encounter a less flexible variant of **ps**. The **ps** manpage can be, depending on the distribution being questioned, up to nearly 900 lines long. Because of its versatile nature, you are encouraged to read through the manpage and try out some of the options **ps** has to offer.

Examples:

```
$ ps ef
PID TTY      STAT   TIME COMMAND
4417 pts/0    Ss     0:00 bashDISPLAY=:0 PWD=/home/user HOME=/home/user SESSI
4522 pts/0    R+     0:00  \_ ps efSSH_AGENT_PID=4206 GPG_AGENT_INFO=/home/user/

$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 02:02 ?        00:00:01 init [2]
root         2     0  0 02:02 ?        00:00:00 [kthreadd]
root         3     2  0 02:02 ?        00:00:01 [ksoftirqd/0]
root         4     2  0 02:02 ?        00:00:01 [kworker/0:0]
root         6     2  0 02:02 ?        00:00:00 [migration/0]
root         7     2  0 02:02 ?        00:00:00 [watchdog/0]
```

pstree

The **pstree** command shows the same processes as **ps** and **top**, but the output is formatted as a tree. The tree is rooted at pid (or **init** if pid is omitted), and if a username is specified the tree will root at all processes owned by that username. **pstree** provides an easy way to track back a process to its parent process id (PPID). Output between square brackets prefixed by a number are identical branches of processes grouped together, the prefixed number represents the repetition count. Grouped child threads are shown between square brackets as well but the process name will be shown between curly braces as an addition. The last line of the output shows the number of children for a given process.

Usage:

```
$ pstree [options] [pid|username]
```

Example:

```
$ pstree 3655
gnome-terminal---bash---pstree
               |-bash---man---pager
               |-gnome-pty-helpe
               `-3*[{gnome-terminal}]
```

w

s The **w** command displays information about the users currently logged on to the machine, their processes and the same statistics as provided by the **uptime** command.

Usage:

```
$ w [options] [user]
```

Example:

```
$ w -s
02:52:10 up 49 min,  2 users,  load average: 0.11, 0.10, 0.13
USER     TTY      FROM              IDLE WHAT
user     tty9     :0                49:51  gdm-session-worker [pam/gdm3]
user     pts/0    :0                0.00s w -s
```

Option `-s` stands for “short format”.

lsof

The **lsof** command is used to list information about open files and their corresponding processes. **lsof** will handle regular files, directories, block special files, character special files, executing text references, libraries, streams or network files. By default, lsof will show unformatted output which might be hard to read but is very suitable to be interpreted by other programs. The `-F` option plays an important role here. The `-F` option is used to get output that can be used by programs like C, Perl and awk. Read the manpages for detailed usage and the possibilities.

Usage:

```
$ lsof [options] [names]
```

`names` acts as a filter here, without options **lsof** will show *all* open files belonging to *all* active processes.

Examples:

```
$ sudo lsof /var/run/utmp
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
gdm-simpl 4040 root   10u   REG   0,14     5376  636 /run/utmp

$ sudo lsof +d /var/log
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
rsyslogd 2039 root    1w   REG    8,1    44399 262162 /var/log/syslog
rsyslogd 2039 root    2w   REG    8,1   180069 271272 /var/log/messages
rsyslogd 2039 root    3w   REG    8,1    54012 271269 /var/log/auth.log
rsyslogd 2039 root    6w   REG    8,1   232316 271268 /var/log/kern.log
rsyslogd 2039 root    7w   REG    8,1   447350 271267 /var/log/daemon.log
rsyslogd 2039 root    8w   REG    8,1    68368 271271 /var/log/debug
rsyslogd 2039 root    9w   REG    8,1     7888 271270 /var/log/user.log
Xorg     4041 root    0r   REG    8,1    31030 262393 /var/log/Xorg.0.log
```

This last example causes **lsof** to search for all open instances of directory `/var/log` and the files and directory it contains at its top level.

free

The **free** command shows a current overview of the total amount of both physical and virtual memory of a system, as well as the amount of free memory, memory in use and buffers used by the kernel.

The fourth column, called *shared* has been obsolete but is now used to display the memory used for tmpfs (shmem in /proc/meminfo)

Usage:

```
$ free  [options]
```

Example:

```
$ free -h
             total       used       free     shared    buffers     cached
Mem:          502M       489M        13M        50B        44M       290M
-/+ buffers/cache:       154M       347M
Swap:         382M       3.9M       379M
```

top

The **top** command provides a “dynamic real-time view” of a running system.

Usage:

```
$ top [options]
```

Example:

```
$ top
top - 03:01:24 up 59 min,  2 users,  load average: 0.15, 0.19, 0.16
Tasks: 117 total,   2 running, 115 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.9 us,  4.5 sy,  0.1 ni, 94.3 id,  0.1 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem:    514332 total,   497828 used,    16504 free,    63132 buffers
KiB Swap:   392188 total,        0 used,   392188 free,   270552 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 4041 root      20   0  106m  31m 9556 R  30.4  6.3   3:05.58 Xorg
 4262 user      20   0  527m  71m  36m S  18.2 14.3   2:04.42 gnome-shell
```

Because of its interactive mode, the most important keys while operating **top** are the *help* keys **h** or **?** and the *quit* key **q**. The following scheme provides an overview of the most important function keys and it's alternatives:

```
key      equivalent-key-combinations
Up       alt + \      or  alt + k
Down     alt + /      or  alt + j
Left     alt + <      or  alt + h
Right    alt + >      or  alt + l (lower case L)
PgUp     alt + Up     or  alt + ctrl + k
PgDn     alt + Down   or  alt + ctrl + j
Home     alt + Left   or  alt + ctrl + h
End      alt + Right  or  alt + ctrl + l
```

htop

Usage:

```
$ htop [options]
```

The **htop** command is similar to the **top** command, but allows you to scroll vertically and horizontally, so you can see all the processes running on the system, along with their full command lines. Tasks related to processes (killing, renicing) can be done without entering their PIDs.

Example:

```
$ htop
```

```
  1  [|                          2.0%]   Tasks: 138, 445 thr; 1 running
  2  [|                          0.7%]   Load average: 0.46 0.19 0.10
  3  [                           0.0%]   Uptime: 05:07:28
  4  [                           0.0%]
  Mem[||||||||             2.29G/15.5G]
  Swp[                        0K/7.84G]

 USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 root       20   0  771M 48796 10560 S  0.7  0.3  0:00.81 /usr/libexec/fwupd/fwupd
 root       20   0  276M  141M  141M S  0.0  0.9  0:21.70 /usr/lib/systemd/systemd-jour
 root       20   0  683M 15696 11916 S  0.0  0.1  0:17.02 /usr/sbin/NetworkManager --nc
 root       20   0  683M 15696 11916 S  0.0  0.1  0:03.10 /usr/sbin/NetworkManager --nc
 root       20   0  264M  5704  4760 S  0.0  0.0  0:05.93 /usr/sbin/iio-sensor-proxy
 root       20   0 66408  8152  7208 S  0.0  0.1  0:01.20 /usr/sbin/wpa_supplicant -c /
 root       20   0  339M 11880 10564 S  0.0  0.1  0:01.36 /usr/bin/abrt-dump-journal-xc
 root       20   0  4416   652   588 S  0.0  0.0  0:00.83 /sbin/rngd -f
 root       20   0  370M 16888 15720 S  0.0  0.1  0:01.25 /usr/bin/abrt-dump-journal-oc
 root       20   0  771M 48796 10560 S  0.0  0.3  0:00.32 /usr/libexec/fwupd/fwupd
 root       20   0  526M 18980 15376 S  0.0  0.1  0:01.62 /usr/libexec/packagekitd
 root       20   0  209M 10616  7512 S  0.0  0.1  0:01.77 /usr/lib/systemd/systemd --sv
 root       20   0 47796  7364  4836 S  0.0  0.0  0:00.24 /usr/lib/systemd/systemd-udev
 root       16  -4 55552  3436  3068 S  0.0  0.0  0:00.00 /sbin/auditd -n
 root       16  -4 55552  3436  3068 S  0.0  0.0  0:00.02 /sbin/auditd -n
 root       12  -8 84544  1788  1652 S  0.0  0.0  0:00.02 /sbin/audispd
 root       12  -8 84544  1788  1652 S  0.0  0.0  0:00.03 /sbin/audispd
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

*The **htop** window.*

uptime

The **uptime** command shows how long the system has been running, how many users are logged on, the system load averages for the past 1, 5 and 15 minutes and the current time. It support the `-V` option for version information.

Usage:

```
$ uptime [options]
```

Example:

```
$ uptime
03:03:12 up  1:00,  2 users,  load average: 0.17, 0.18, 0.16
```

sar

The **sar** command collects, reports or saves system activity information.

Usage:

```
$ sar [options] [interval [count] ]
```

Examples:

```
$ sar
Linux 3.2.0-4-686-pae (debian)  05/07/2013  _i686_  (2 CPU)

02:02:34      LINUX RESTART
```

```
02:05:01     CPU     %user     %nice   %system   %iowait    %steal     %idle
02:15:01     all      0.15      0.00      1.06      0.23      0.00     98.56
02:25:01     all      0.98      0.83      3.84      0.04      0.00     94.31
02:35:01     all      0.46      0.00      4.84      0.04      0.00     94.66
02:45:01     all      0.90      0.00      5.29      0.01      0.00     93.80
02:55:01     all      0.66      0.00      4.64      0.03      0.00     94.67
03:05:02     all      0.66      0.00      5.57      0.01      0.00     93.76
Average:     all      0.64      0.14      4.19      0.06      0.00     94.98
```

Without options, sar will output the statistics above.

Using the `-d` option sar will output disk statistics.

```
$ sar -d
06:45:01      DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz avgqu-sz   await   svctm   %util
06:55:01   dev8-0      6.89    227.01     59.67     41.59     0.02    2.63    1.38    0.95
07:05:01   dev8-0      2.08     17.73     17.78     17.06     0.00    2.19    0.94    0.20
07:15:01   dev8-0      1.50     12.16     12.96     16.69     0.00    1.35    0.68    0.10
Average:   dev8-0      3.49     85.63     30.14     33.15     0.01    2.36    1.19    0.42
```

The `-b` option switch shows output related to I/O and transfer rate statistics :

```
$ sar -b
06:45:01       tps      rtps      wtps   bread/s   bwrtn/s
06:55:01      6.89      4.52      2.38    227.01     59.67
07:05:01      2.08      0.95      1.13     17.73     17.78
07:15:01      1.50      0.50      1.00     12.16     12.96
Average:      3.49      1.99      1.50     85.63     30.14
```

Some of the most important options to be used with **sar** are:

- `-c` System calls
- `-p and -w` Paging and swapping activity
- `-q` Run queue
- `-r` Free memory and swap over time

Match / correlate system symptoms with likely problems

In order to be able to troubleshoot a certain problem, one must first be able to distinguish normal system behaviour from abnormal system behaviour.

In the previous section, a number of very specific system utilities as well as their utilization is explained. In this section, the focus lies on correlating these measurements and being able to detect anomalies, which in turn can be tied to abnormal system behaviour. Resource related problems have a very distinguishable factor in common:

They are the result of one or more resources not being able to cope with the demand during certain circumstances.

These resources might be related, but are not limited to: the CPU, physical or virtual memory, storage, network interfaces and connections, or the input/output between one or more of these components.

Estimate throughput and identify bottlenecks in a system including networking

To determine whether or not a certain problem is related to a lack of resources, the problem itself has to be properly formulated first. Then, this formulated "deviated" behaviour has to be compared to the expected behaviour which would result from a trouble-free operating system.

If possible, historical data from **sar** or other tools should be investigated and compared to real-time tools like **top**, **vmstat**, **netstat** and **iostat**.

Problems reported by either users or reporting tools are often related to availability , either certain resources aren't available in an orderly fashion or not at all. Orderly fashion might be interpreted as "within an acceptable period of time" here. These kind of issues are often reported because they are very noticeable for users, it affects their *user experience*.

Examples of these kind of issues might be certain files or (web)applications which aren't accessible or responding in a certain period of time. To adequately analyse the situation, it would be handy to have a *baseline* at hand which dictates what the "expected behaviour" should be. This baseline should be determined on a system which behaves properly, and a certain amount of threshold should be considered.

If there is no baseline, the measurements themselves should be able to help determine the root cause of a resource related problem. If one of the resources mentioned above is at 100% of it's capacity, the reason for the abnormal system behaviour should be easy to explain. Finding the cause however takes a bit more effort. The utilities presented in the previous chapter however, should be able to help out here as well.

Examples:

```
$ top
$ vmstat
$ iostat
$ netstat
```

Identifying bottlenecks in a networking environment requires several steps. A best practice approach could be outlined as follows:

- Create a map of the network
- Identify time-dependent behaviour
- Identify the problem
- Identify deviating behaviour
- Identify the cause

Predict Future Resource Needs (200.2)

Candidates should be able to to predict future resource needs.

Key Knowledge Areas:

- Use monitoring and measurement tools to monitor IT infrastructure usage
- Predict capacity break point of a configuration
- Observe growth rate of capacity usage
- Graph the trend of capacity usage.
- Awareness of monitoring solutions such as Icanga2, Nagios, collectd, MRTG and Cacti.

The following is a partial list of used files, terms and utilities:

- diagnose
- predict growth
- resource exhaustion

Using the tools and knowledge presented in the previous two chapters, it should be possible to diagnose the usage of resources for specific components or processes. One of the tools mentioned was sar, which is capable of recording measurements over a longer period of time. Being able to use the recorded data for trend analysis is one of the advantages of using a tool that is able to log measurements.

Monitor IT infrastructure

One of the tools that can be used to monitor an IT infrastructure is *collectd*. collectd is a daemon which collects system performance statistics periodically and provides mechanisms to store the values in a variety of ways. It gathers statistics about the system it is running on and stores this information. Those statistics can then be used to find current performance bottlenecks (i.e. performance analysis) and predict future system load (i.e. capacity planning). collectd is written in C for performance and portability, allowing it to run on systems without scripting language or cron daemon, such as embedded systems. Note that collectd only collects data, to display the collected data addtional tools are required.

Also other monitoring tools like Icanga2, Nagios, MRTG and Cacti can be used to measure, collect and display resource performance statistics. Using a montoring tool can help you identify possible bottlenecks related to the measured resources but can also help you predict future growth.

Predict future growth

By analyzing and observing the data from measurements, over time it should be possible to predict the *statistical* growth of resource needs. We deliberately say statistical growth here, because there are many circumstances which can influence resource needs. The demand for fax machines and phone lines has suffered from the introduction of e-mail, for instance. But numerical or statistical based growth estimations also suffer from the lack of linearity: When expanding due to increased demand the expansion often incorporates a variety of services. The demand for these services usually doesn't grow at the same speed for all provided services. This means that measurement data should not just be analysed, but also evaluated regarding to judge its relevance.

The steps to predict future needs can be done as follows:

- Decide what to measure.
- Use the appropriate tools to measure and record relevant data to meat your goals.
- Analyze the measurement results, starting with the biggest fluctuations.
- Predict future needs based on the analysis.

Resource Exhaustion

When a resource cannot deliver to the request in an orderly fashion anymore, it is exhausted. The demand and delivery are not aligned anymore, and the availability of resources will become a problem. Resource Exhaustion can lead to a denial-of-service. Apart from disrupting the availability of a resource, devices which are configured to "fail open" can be tricked by exhausting it's resources. Some switches fall back to forwarding all traffic to all ports when the ARP table becomes flooded, as an example.

Most of the time, a single resource which gets exhausted will be extractable from collected measurement data. This is what we call a *bottleneck*: a single point within the system narrows throughput and slows down everything below. It is important to have a clear understanding of the bigger picture here. Simply resolving a specific bottleneck will only shift the problem, if you increase the capacity of one component another component will become the limiting factor as soon as it hits it's capacity limit.

Therefore it is important to identify as many bottlenecks as possible right away during analysis.

Questions and answers

Capacity Planning

1. *What tools can be used to show block device Input/Output statistics?*

   iostat, **vmstat**, and **sar** are some of the tools that can be used to output block device I/O. The iostat command [2] The vmstat command [3] The sar command [10]

2. *What does the first column of the output of vmstat, called 'r' represent?*

   The number of processes currently allocated to the processor run queue. processor run queue [4]

3. *Which kernel module has to be loaded on post-2.4 kernels in order for /proc/net/ip_conntrack to exist?*

   Only if the ipt_MASQUERADE kernel module is loaded will /proc/net/ip_conntrack exist. IP Masquerading [5]

4. *What is the main difference between **netstat -r** and **route -e** since their output is identical?*

   The **route -e** requires elevated privileges whereas **netstat -r** can be run with user privileges. netstat --route or -r option [5]

5. *What are the three different kind of options which can be passed to the GNU **ps** command?*

   GNU long options, UNIX and BSD. The ps command [6]

6. *What command can be used to display processes formatted as a tree?*

   pstree The pstree command [7]

7. *What command can be used to list all open files and their corresponding processes?*

   The **lsof** will list all open files and their corresponding processes. The lsof command [8]

8. *Resource related problems are the result of what?*

   Resource related problems are the result of one or more resources not being able to cope with the demand during certain circumstances. Resource Exhaustion [13]

9. *Problems reported by users are often related to availability or resources, and the performance of their application in particular. What needs to be determined first in order to be able to determine whether these lack of performance based claims are indeed valid or not?*

   A baseline needs to be determined on a system which behaves properly. This baseline can then be used as a reference for identical or similar systems. Create a baseline [12]

10. *What needs to be created first when investigating a possible network related bottleneck?*

    A map of the network should be created before troubleshooting network issues. Identifying network bottlenecks [12]

11. *What command can be used to show both the amount of physical and virtual memory, and how much memory is being used?*

    Several utilities like **top** and **htop** can be used, but the **free** command will show detailed information about the allocation of the memory in use. (**htop** is an interactive process viewer from the Debian htop package) The free command [8]

Chapter 1

Linux Kernel (201)

This topic has a total weight of 9 points and contains the following three objectives:

Objective 201.1: Kernel Components (weight: 2) Candidates should be able to utilize kernel components that are necessary to specific hardware, hardware drivers, system resources and requirements. This objective includes implementing different types of kernel images, understanding stable and longterm kernel kernels and patches, as well as using kernel modules.

Objective 201.2: Compiling a Linux kernel (weight: 3) 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 201.3: Kernel runtime management and troubleshooting (weight: 4) Candidates should be able to manage and/or query a 2.6.x, 3.x or 4.x kernel and its loadable modules. Candidates should be able to identify and correct common boot and run time issues. Candidates should understand device detection and management using udev. This objective includes troubleshooting udev rules.

Kernel Components (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 knowledge Areas:

- Kernel 2.6.x documentation
- Kernel 3.x documentation
- Kernel 4.x documentation

Terms and Utilities

- `/usr/src/linux`
- `/usr/src/linux/Documentation`
- `zImage`
- `bzImage`
- `xz compression`

Resources: wikiXZ, tukaani;

Different types of kernel images

The Linux kernel was originally designed to be a monolithic kernel. Monolithic kernels contain all drivers for all the various types of supported hardware, regardless if your system uses that hardware. As the list of supported hardware grew the amount of code that was never used on any given system grew too. Therefore a system was introduced that allowed the kernel to load some hardware drivers dynamically. These loadable device drivers were named "kernel modules".

Though the Linux kernel can load and unload modules it does not qualify as a microkernel. Microkernels are designed such that only the least possible amount of code is run in supervisor mode – this was never a design goal for Linux kernels. The Linux kernel is best described as a hybrid kernel: it is capable of loading and unloading code as microkernels do, but runs almost exclusively in supervisor mode, as monolithic kernels do.

It is still possible to build the Linux kernel as a monolithic kernel. But it is rarely done, as updating device drivers requires a complete recompile of the kernel. However, building a monolithic kernel has its advantages too: it may have a smaller footprint as you can download and build just the parts you need and dependencies are clearer.

When stored on disk most kernel images are compressed to save space. There are two types of compressed kerneltypes: `zImage` and `bzImage`.

`zImage` and `bzImage` files have different layouts and loading algorithms. The maximum allowed kernelsize for a `zImage` is 512Kb, where a `bzImage` does not pose this limit. As a result the `bzImage` kernel is the preferred image type for larger kernels. `zImage` will be loaded in low memory and `bzImage` can also be loaded in high memory if needed.

Note
Both `zImage` and `bzImage` use gzip compression. The “bz” in `bzImage` refers to “*big zImage*” – not to the “bzip” compression algorithm.

Overview of numbering schemes for kernels and patches

The numbering schemes in use for Linux kernels has changed several times over the years: the original scheme, valid for all kernels up to version 2.6.0, the scheme for kernels version 2.6.0 up to 3.0, the previous scheme, for kernels 3.0 and later, and the current scheme starting with version 4.0. In the next sections we discuss each of them.

Scheme up to 2.6.0 kernels

Initially, a kernel version number consisted of three parts: major release number, minor release number and the patch level, all separated by periods.

The major release was incremented when a major change was made to the kernel.

The minor release was incremented when significant changes and additions were made. Even-numbered minor releases, e.g. 2.2, 2.4, were considered stable releases and odd-numbered releases, e.g. 2.1, 2.3, 2.5 were considered to be development releases. They were primarily used by kernel developers and people that preferred bleeding edge functionality at the risk of instability.

The last part of the kernel version number indicated the patch level. As errors in the code were corrected (and/or features were added) the patch level was incremented. A kernel should only be upgraded to a higher patch level when the current kernel has a functional or security problem.

Kernel Versioning since kernel version 2.6.0 and up to 3.0

In 2004, after the release of 2.6.0, the versioning system was changed, and it was decided that a time-based release cycle would be adopted. For the next seven years the kernel remained at 2.6 and the third number was increased with each new release (which happend every two or three months). A fourth number was added to account for bug and security fixes. An example of this scheme is kernel `2.6.32.71`. The even-odd numbering system was no longer used.

Kernel Versioning from version 3.0 to 4.0

On 29 May 2011, Linus Torvalds announced the release of kernel version 3.0.0 in honour of the 20th anniversary of Linux. This changed the numbering scheme yet again. It would still be a time-based release system but the second number would indicate the release number, and the third number would be the patch number. For test releases the -rc designation is used. Following this scheme, `3.2.84` would refer to a stable kernel release. `3.2-rc4` on the other hand would point to the fourth release candidate of the `3.2` kernel.

Kernel Versioning from 4.0

In April 2015 kernel version 4.0.0 was released. The versioning system didn't change this time. At the time of this writing, kernel version `4.9.2` is the latest stable version available through https://kernel.org. The 4.x kernel did however introduce a couple of new features. The possibility to perform "Live Patching" being one of the more noteworthy ones. Live patching offers the possibility to install kernel patches without the need to reboot the system. This can be accomplished by unloading and loading appropriate kernel modules. Every time a new Linux kernel version gets released, it is accompanied by a `changelog`. These changelog files hold detailed information about what has changed in this release compared to previous versions.

XZ Compression

Every Linux distribution comes with a kernel that has been configured and compiled by the distribution developers. Most Linux distributions also offer possibilities to upgrade the kernel binary through some sort of package system. It is however also possible to compile a kernel for your system using kernel sources from the previously mentioned website kernel.org. These kernel sources are packed using tar and compressed using the *XZ* compression method. XZ is the successor to LZMA and LZMA2. Recent Linux kernels offer built-in support for XZ. Depending on the Linux distribution in use, it might be necessary to install a `xz-utils` or equivalent package to uncompress xz compressed files. After having downloaded the kernel sources for one of the available kernels as a `tar.xz` archive, these source files may be unpacked using the following command line:

```
$ tar xvf linux-4.10-rc3.tar.xz
```

Note
GNU **tar** needs to be at least version 1.22 for the above command to work.

What are kernel modules

Linux kernel modules are object files (`.ko` files) 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** or found in `/proc/sys/kernel/osrelease`, such as 2.6.5-15smp. Multiple module hierarchies are available under `/lib/modules` in case multiple kernels are installed.

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

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.ko module).

ipv4 Includes modular kernel features having to do with IP processing, such as IP masquerading.

misc Anything that does not 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 within the context of module manipulation commands.

Compiling a Linux kernel (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 Knowledge Areas

- `/usr/src/linux/`
- Kernel Makefiles
- Kernel 2.6.x, 3.x and 4.x make targets
- Customize the current kernel configuration
- Build a new kernel and appropriate kernel modules
- Install a new kernel and any modules
- Ensure that the boot manager can locate the new kernel and associated files
- Module configuration files
- Use DKMS to compile kernel modules
- Awareness of **dracut**

Terms and Utilities

- **mkinitrd**
- **mkinitramfs**
- **make**
- **make** targets (all, config, xconfig, menuconfig, gconfig, oldconfig, mrproper, zImage, bzImage, modules, modules_install, rpm-pkg, binrpm-pkg, deb-pkg)
- **gzip**
- **bzip2**
- module tools
- `/usr/src/linux/.config`
- `/lib/modules/kernel-version/`
- **depmod**

Resources: archDKMS, debianDKMS; wikiDKMS; githubDKMS; dracut;

Getting the kernel sources

Kernel sources for almost all kernel versions can be found at *The Linux Kernel Archives*.

The filenames in the Linux Kernel Archive mimic the version numbering conventions for the kernel. For example: The filename format for kernel version 3.0 and 4.0 is `linux-`*kernel-version*`.tar.xz` Thus, `linux-3.18.43.tar.xz` is the kernel archive for version "3.18.43".

The used version numbering convention for the 3.0 and 4.0 kernel is `linux-`*A.B.C*`.tar.xz` where:

- A denotes the kernel version. It is only changed when major changes in code and concept take place.
- B denotes the revision.
- C is the patch number

See the paragraph on Kernel Versioning to learn more about the various conventions that are and have been in use.

A common location to store and unpack kernel sources is `/usr/src`. You can use another location as long as you create a symbolic link from your new source directory to `/usr/src/linux`

The source code for the kernel is available as a compressed tar archive in **xz** (`.xz` extention) format. Decompress the archive with **unxz**. The resulting **tar** archive can be unpacked with the **tar** utility, for example:

```
# unxz linux-3.18.43.tar.xz
# tar xvf linux-3.18.43.tar
```

You can also uncompress and untar in one step **tar** using the *J*option:

```
# tar Jxvf linux-3.18.43.tar.xz
```

Refer to the man-pages on **tar** and, **xz** for more information.

Cleaning the kernel

To make sure you start with a clean state you should "clean" the kernel first. When you compile a kernel into objects, the **make** utility keeps track of things and will not recompile any code it thinks has been correctly compiled before. In some cases, however, this may cause problems, especially when you change the kernel configuration. It is therefore customary to "clean" the source directory if you reconfigure the kernel.

Cleaning can be done on three levels:

make *clean* Deletes most generated files, but leaves enough to build external modules.

make *mrproper* Deletes the current configuration and all generated files.

make *distclean* Removes editor backup files, patch leftover files and the like.

Running **make** *mrproper* before configuring and building a kernel is generally a good idea.

Note
Be warned that **make** *mrproper* deletes the main configuration file too. You may want to make a backup of it first for future reference.

Creating a `.config` file

First you will need to configure the kernel. Configuration information is stored in the `.config` file. There are well over 500 options in that file, for example for filesystem, SCSI and networking support. Most of the options allow you to choose if you will have them compiled directly into the kernel or have them compiled as a module. 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.

Some of the kernel support options *must* be compiled as a module, some can only be compiled as permanent part of the kernel and for some options you will be able to select either possibility.

There are a number of methods to configure the kernel, but regardless which method you use, the results of your choices are always stored in the kernel configuration file `/usr/src/linux/.config`. It is a plain text file which lists all the options as shell variables.

Example 1.1 Sample `.config` content

```
#
# Automatically generated make config: don't edit
# Linux kernel version: 2.6.28
# Sat Feb  6 18:16:23 2010
#
CONFIG_64BIT=y
# CONFIG_X86_32 is not set
CONFIG_X86_64=y
CONFIG_X86=y
CONFIG_ARCH_DEFCONFIG="arch/x86/configs/x86_64_defconfig"
CONFIG_GENERIC_TIME=y
CONFIG_GENERIC_CMOS_UPDATE=y
CONFIG_CLOCKSOURCE_WATCHDOG=y
CONFIG_GENERIC_CLOCKEVENTS=y
CONFIG_GENERIC_CLOCKEVENTS_BROADCAST=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_HAVE_LATENCYTOP_SUPPORT=y
CONFIG_FAST_CMPXCHG_LOCAL=y
CONFIG_MMU=y
CONFIG_ZONE_DMA=y
CONFIG_GENERIC_ISA_DMA=y
CONFIG_GENERIC_IOMAP=y
CONFIG_GENERIC_BUG=y
CONFIG_GENERIC_HWEIGHT=y
...
```

To start configuration, change your current working directory to the top of the source tree:

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

As said, there are several ways to create or modify the `.config` file. It is strongly discouraged to edit this file manually. Instead you should use the **make** command with one of the four appropriate targets to configure your kernel.

These four targets are:

- config
- menuconfig
- xconfig|gconfig
- oldconfig

These targets will be explained below in more detail.

make *config*

Running **make** *config* is the most rudimentary approach.

It has clear advantages and disadvantages:

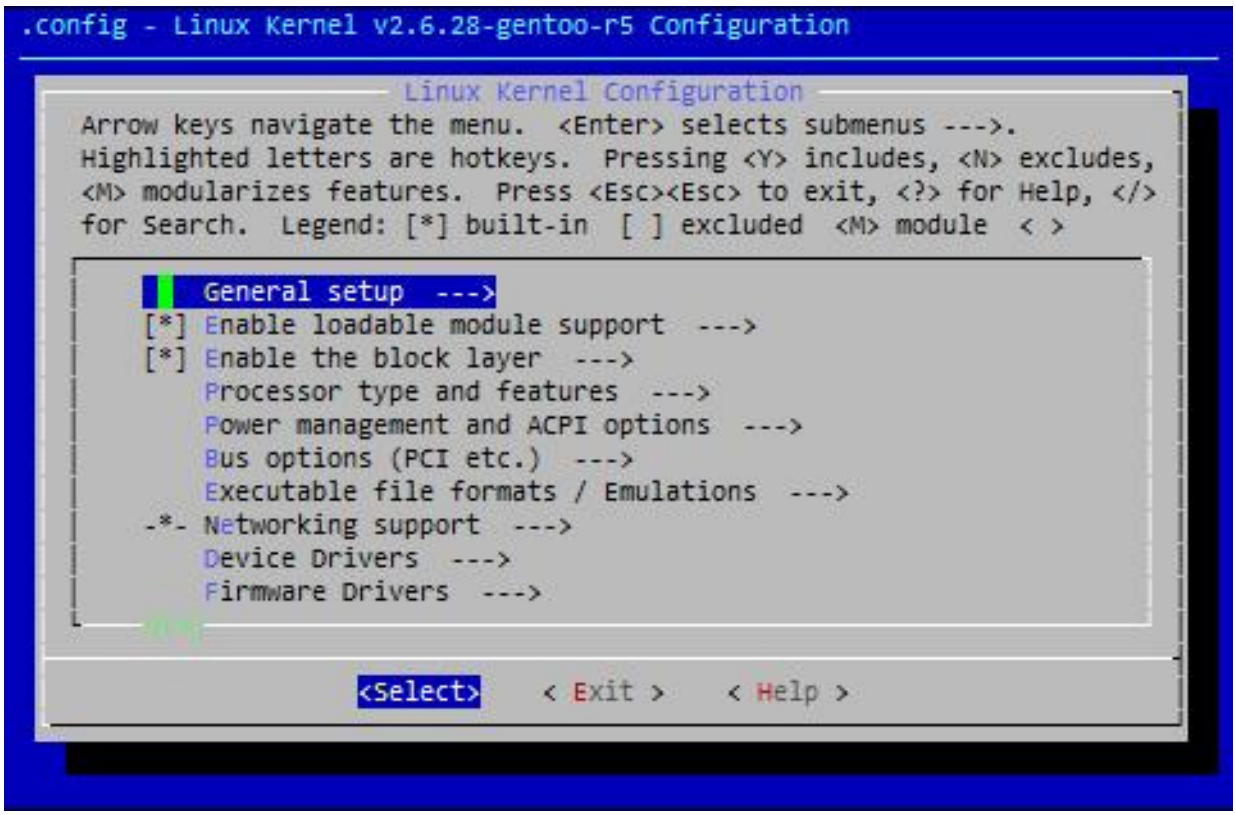
- It does not depend on full-screen display capabilities. You can use it on extremely slow links, or on systems with very limited display capabilities.
- You will have to work your way through *all* possible questions concerning kernel options. The system will present them sequentially and without exception. Only when you have answered *all* questions will you be allowed to save the configuration file. Given that, there are many hundreds of options to go through so this method is tedious. Because you cannot move back and forth through the various questions you are forced to redo everything if you make a mistake.

An example session looks like this:

```
# make config
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/basic/docproc
  HOSTCC  scripts/basic/hash
  HOSTCC  scripts/kconfig/conf.o
scripts/kconfig/conf.c: In function 'conf_askvalue':
scripts/kconfig/conf.c:104: warning: ignoring return value of 'fgets', \
    declared with attribute warn_unused_result
scripts/kconfig/conf.c: In function 'conf_choice':
scripts/kconfig/conf.c:306: warning: ignoring return value of 'fgets', \
    declared with attribute warn_unused_result
  HOSTCC  scripts/kconfig/kxgettext.o
  HOSTCC  scripts/kconfig/zconf.tab.o
In file included from scripts/kconfig/zconf.tab.c:2486:
scripts/kconfig/confdata.c: In function 'conf_write':
scripts/kconfig/confdata.c:501: warning: ignoring return value of 'fwrite', \
    declared with attribute warn_unused_result
scripts/kconfig/confdata.c: In function 'conf_write_autoconf':
scripts/kconfig/confdata.c:739: warning: ignoring return value of 'fwrite', \
    declared with attribute warn_unused_result
scripts/kconfig/confdata.c:740: warning: ignoring return value of 'fwrite', \
    declared with attribute warn_unused_result
In file included from scripts/kconfig/zconf.tab.c:2487:
scripts/kconfig/expr.c: In function 'expr_print_file_helper':
scripts/kconfig/expr.c:1090: warning: ignoring return value of 'fwrite', \
    declared with attribute warn_unused_result
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf arch/x86/Kconfig
*
* Linux Kernel Configuration
*
*
* General setup
*
Prompt for development and/or incomplete code/drivers (EXPERIMENTAL) [Y/n/?]
```

make *menuconfig*

The **make** *menuconfig* method is more intuitive and can be used as an alternative to **make** *config*. It creates a text-mode windowed environment based on the **ncurses** libraries. You can switch back and forth between options. The sections are laid out in a menu-like structure which is easy to navigate and you can save and quit whenever you want. If you prefer a darker color scheme, use **make** *nconfig*.



*The **make** `menuconfig` menu display.*

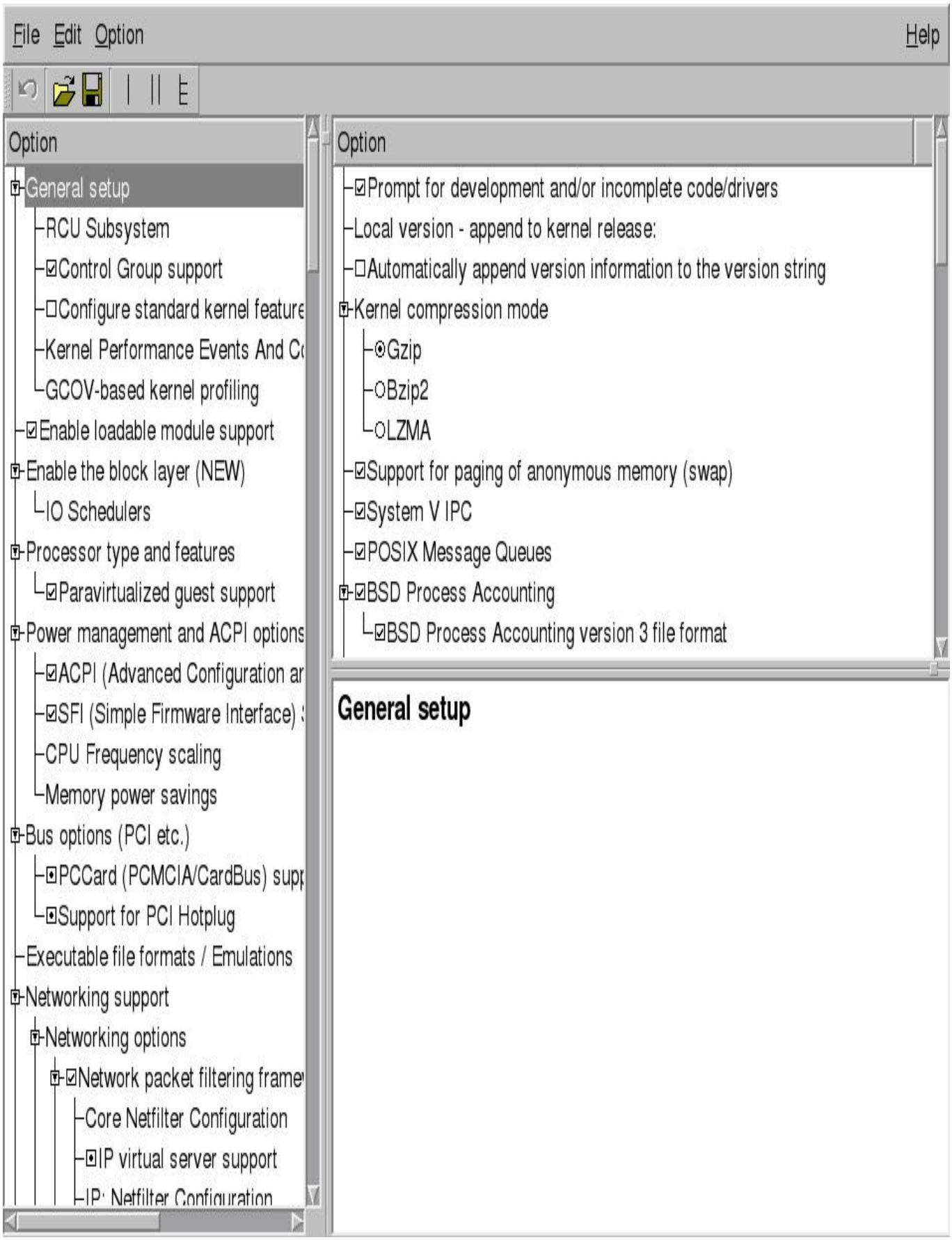
When done, use the arrow keys to select the *Exit* option at the bottom of the screen. If any changes were made you will be prompted if you would like to save the new configuration. You can also choose to save the configuration using another name and/or location in the filesystem.

Note
If you choose another name or location you need to move the `.config` file into the `/usr/src/linux` directory to compile the kernel.

---

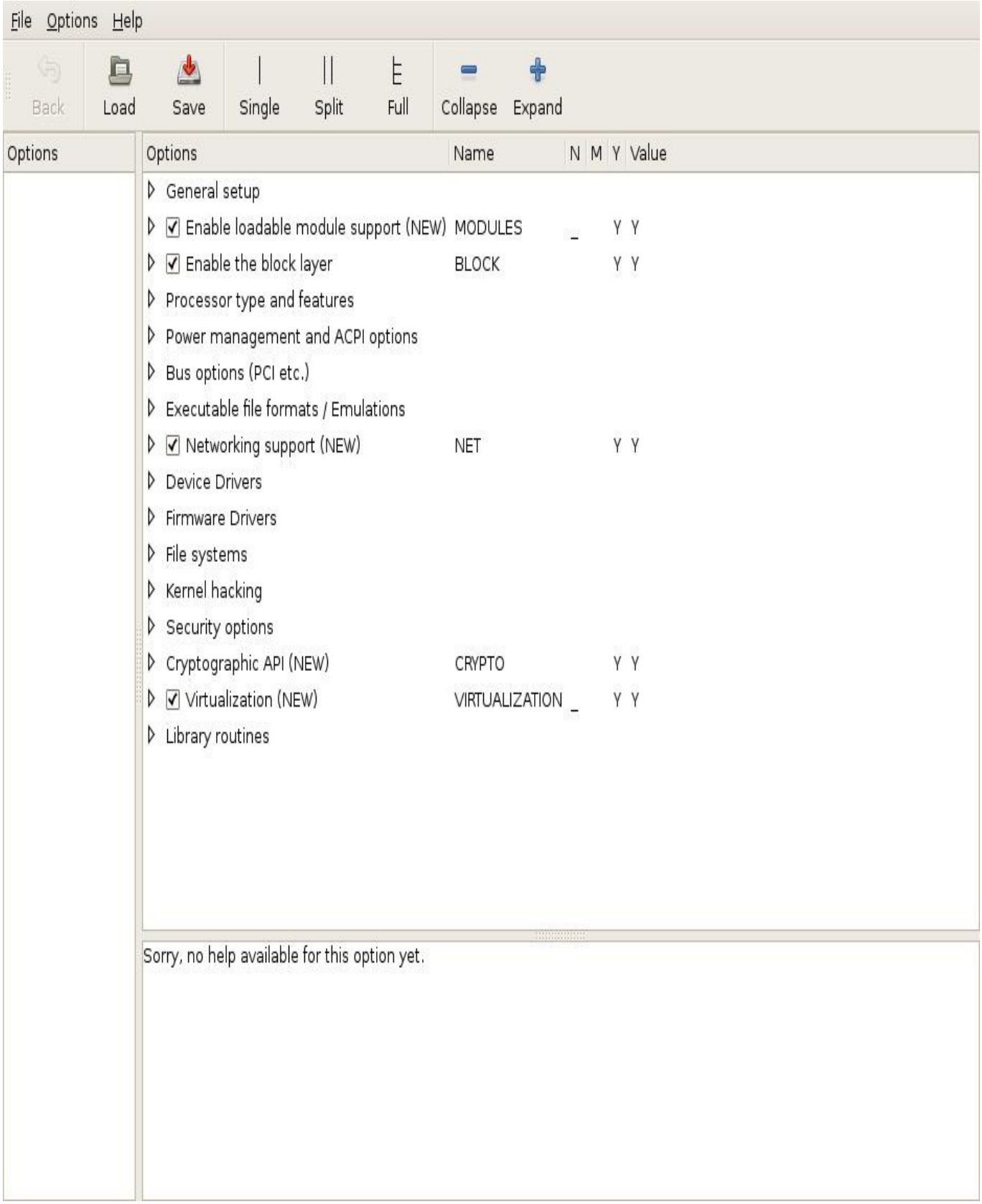
make *`xconfig`* and *`gconfig`*

The **make** *`xconfig`* command presents a GUI menu to configure the kernel. It requires a working X Window System and the QT development libraries to work. It will provide a menu which can be navigated using a mouse. Use **make** *`gconfig`* to use Gnome instead of QT. This requires the GTK+ 2.x development libraries to be available. First, we show you how the top-level **make** *`xconfig`* window looks:



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

As said, the command **make** *gconfig* does exactly the same, but uses GTK instead of QT:



*The **make** `gconfig` top-level window.*

make *oldconfig*

make *oldconfig* can be used to preserve options you choose during an earlier kernel build.

Make sure the `.config` file that was the result of the earlier build is copied into `/usr/src/linux/`. When you run **make** *oldconfig*, the original `.config` file will be moved to `.config.old` and a new `.config` will be created. You will be prompted for answers that can not be found in the previous configuration file, for example when there are new options for the new kernel.

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*, **make** *gconfig* and **make** *menuconfig* will *automatically* use the old `.config` file (if available) to construct a new one, preserving as much options as possible while adding new options using their default values.

Compiling the kernel

Use the following sequence of **make** commands to build and install the kernel and modules:

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

make *clean*

The "clean" argument 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** arguments both effectively build the kernel. The difference between these two is explained in zImage versus bzImage [16].

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

make *modules*

The **modules** argument builds the modules; the device drivers and other items that were configured as modules.

make *modules_install*

The **modules_install** argument installs the modules you just compiled under `/lib/modules/`*kernel-version*. The *kernel-version* directory will be created if nonexistent.

Installing the new kernel

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

First you need to put a copy of the new `bzImage` in the boot directory (which should reside on its own boot partition). For clarity the name of the kernel file should contain the kernel-version number, for example: `vmlinuz-2.6.31`:

```
# cp /usr/src/linux/arch/x86_64/boot/bzImage
/boot/vmlinuz-2.6.31
```

This also ensures that you can have more than one kernel version in the `/boot` directory, for example if you need to boot an older kernel due to problems with the new one.

After moving the kernel file to the correct location, you will need to configure the bootmanager (GRUB) so it will be able to boot the new kernel.

For more specific information on GRUB, please refer to GRUB.

The initial ram disk (`initrd`)

Say your bootdisk has the bootloader, kernel and proper modules on it. Given the advantages of kernel modules you decided to use them. But if you also want to use them for the boot device drivers, you face a problem. GRUB will load the kernel, then execute it. The kernel will try to access the disk to obtain the modules. However, as it has not loaded the proper module yet, it can't access that disk and hangs.

A perfectly good solution would be to build a kernel with the required disk-driver hardcoded into it. But if you have a larger number of differing systems to maintain, you either need a personalised configuration and kernel for each type of system or have to live with a bloated kernel. To circumvent all of these problems, the kernel developers came up with a solution: the `initrd` RAM disk.

A RAM disk is a chunk of memory that the kernel sees as if it were a disk. It can be mounted like any other disk. The kernel supports RAM disks by default. GRUB and LILO can handle RAM disks too. You can instruct them to load the RAM disk from a file and when the kernel boots it has the RAM disk readily available. Such RAM disks are often used to hold scripts and modules to aid the boot process.

By convention the name of the image that holds the initial RAM disk is `initrd`. The name is short for "initial ram disk".

The bootloader loads the `initrd`, it is mounted by the kernel as its root filesystem. Once mounted as the root filesystem, programs can be run from it and kernel modules loaded from it. After this step 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.

There are a number of ways to create your own `initrd` file. A very convenient method, mainly used by Red Hat (based) distributions is by using the **mkinitrd** script. It is a shell script which you might want to inspect to see how it works. On Debian–based distributions a utility named **mkinitramfs** can be used for the same purpose. You can also opt to build the file by hand, see the chapter below.

Manual `initrd` creation

`initrd` files are compressed archives that contain the files of a minimal root filesystem. This root filesystem normally contains modules, scripts and some additional binaries required to allow the kernel to properly continue its boot.

As said, the **mkinitrd** script offers a convenient way to build the `initrd` file, however not all distributions provide it. If you want (or must) build one by hand the steps are: create a root filesystem, populate it with modules and files, create a **tar** or **cpio** archive from it and lastly **gzip** it.

What type of archive to use depends on the distribution and kernel version. Older kernels employ **tar**, newer use **cpio**. If you are unsure and have a `initrd` at hand that came with your distribution, you may use a command sequence like the one below to check:

```
$ sudo zcat /boot/initrd-2.6.18-348.6.1.el5.img |file -
/dev/stdin: ASCII cpio archive (SVR4 with no CRC)
```

The example above shows the output of a CentOS 5 distribution that uses **cpio** as its archiving tool.

To be able to work with the `initrd` images, the kernel has to be compiled with support for the RAM disk and configured such that it will use it. Whatever you put on the initial RAM disk, it should be compatible with the kernel and architecture you will use. For example, your boot kernel should be able to recognize the filesystem type used in the image and the modules you include should match the boot kernel version.

The next step is to actually create the RAM disk image. First create a filesystem on a block device and then copy the files to that filesystem as needed. Suitable block devices to be used for this purpose are:

1. A RAM disk (fast, allocates physical memory)
2. A loopback device (slightly slower, allocates disk space)

In the rest of this example we will use the RAM disk method, so we will need to make sure a RAM disk device node is present (there may be more than one):

```
# ls -la /dev/ram0
brw-rw---- 1 root disk 1,  0 Feb 13 00:18 /dev/ram0
```

Note
The number of RAM disks that is available by default on a system is an option in the kernel configuration: CONFIG_BLK_DEV_RAM_COUNT.

Next an empty filesystem needs to be created of the appropriate size:

```
# mke2fs -m0 /dev/ram0 300
```

Note
If space is critical, you may wish to use a filesystem which is more efficient with space, such as the Minix FS. Remember that the boot-kernel will need built-in support for whatever filesystem you choose.

After having created the filesystem, you need to mount it on the appropriate directory:

```
# mount -t ext2 /dev/ram0 /mnt
```

Now the stub for the console device needs to be created. This will be the device node that will be used when the `initrd` is active.

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

Next, copy all files you think are necessary to the image; modules, scripts, binaries, it does not matter. Refer to Contents of /, /boot and /lib/modules [37] for an example of the directories and files needed at minimum. One of the most important files to copy over is `/linuxrc`. Whenever the kernel is set up to use a `initrd` image it will search for a file `/linuxrc` file and execute it. It can be a script or a compiled binary. Hence, what will happen after mounting your image file is totally under your control. In this example we will make `/linuxrc` a link to `/bin/sh`. Make sure `/linuxrc` is given execute permissions.

```
# ln -s /bin/sh /mnt/linuxrc
```

After you have completed copying the files and have made sure that the `/linuxrc` has the correct attributes, you can unmount the RAM disk image:

```
# umount /dev/ram0
```

The RAM disk image can then be copied to a file:

```
# dd if=/dev/ram0 bs=1k count=300 of=/boot/initrd
```

Finally, if you have no more use for the RAM disk and you wish to reclaim the memory, deallocate the RAM disk:

```
# freeramdisk /dev/ram0
```

To test the newly created `initrd`, add a new section to your GRUB menufile, which refers to the `initrd` image you've just created:

```
title=initrd test entry
root (hd0,0)
kernel /vmlinuz-2.6.28
initrd /initrd
```

If you have followed the steps above and have rebooted using this test entry from the bootloader menu, the system will continue to boot. After a few seconds you should find yourself at a command prompt, since `/linuxrc` refers to `/bin/sh`, a shell.

Of course, real `initrd` files will contain a more complex `/linuxrc` boot file, that loads modules, mounts the real root filesystem etc.

Patching a Kernel

Note
This section offers information on a subject that is no longer part of the LPIC-2 objectives. It is maintained because it still contains valid and valuable information.

In older versions of the LPIC-2 objectives candidates were assumed to be able to properly patch the source code of a kernel to add support for new hardware. The objectives included being able to remove kernel patches from patched kernels.

Key files, terms and utilities include:

- Kernel Makefiles
- **patch**
- **xz**

A patch file contains a list of differences between two versions of a file. The standard command **diff** is capable of producing such lists. The command **patch** can be used to apply the contents of a patch file to update the file from the old version to a newer version.

Patching the kernel is very straightforward:

1. Place patch file in the `/usr/src` directory.
2. Change directory to `/usr/src`.
3. Uncompress the patch file using **unxz**
4. Use the **patch** utility to apply the patch file to the kernel source:

   ```
   # patch -p1 <patchfile
   ```

5. Check for failures.
6. Build the kernel.

If the **patch** utility 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 common options for the **patch** utility:

-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 using `-p0` gives the entire file name unmodified, while using `-p1` gives `u/howard/src/blurfl/blurfl.c`.

-s, --silent, --quiet Work silently (suppress output), 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 conforms to the POSIX specification, 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 reversed, so that you are basically applying the patch to the file which already contains the modifications in the patch file. The **patch** will attempt 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. This is 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. In the previous topic we've learned that to remove a patch from a file, you either need to apply it again, or run **patch** with the *-R* parameter:

```
# patch -p1<patch-2.6.28
patching file linux/Documentation/Configure.help
Reversed (or previously applied) patch detected! Assume -R? [n] y
```

DKMS

$ DKMS (Dynamic Kernel Module Support) was developed by Dell in 2003. DKMS was created as a solution to combat software problems caused by the dependencies between kernels and kernel modules. As a vendor of computer systems running (amongst others) Linux operating systems, Dell offered software support to customers. When customers would upgrade the Linux kernel, the kernel modules had to be upgraded as well. And whenever Dell released newer versions of kernel modules for hardware support, these modules had to match the Linux kernel in use.

DKMS is a framework, capable of automatically compiling and/or installing kernel modules for every kernel version available on the system. DKMS achieves this functionality by seperating the kernel module files or sources from the actual kernel source tree. This way, both the kernel and kernel modules may be upgraded independent of each other. Major Linux distributions offer

the DKMS framework through their package system. When the kernel is upgraded by the package manager software on a system running DKMS, a hook wil take care of deciding whether any kernel modules need to be compiled and/or installed for the new kernel. The other way around, new kernel modules can be compiled and/or installed by DKMS without requirements towards the kernel version.

DKMS does have a few requirements in order to function properly. The software dependencies are dependant on the Linux distribution in use. But one requirement that is uniform across all distributions is the necessity for kernel header files. The headers for the running kernel version can be installed using the following package manager commands:

On Red Hat based Linux distributions:

```
$  yum install kernel-devel
```

On Debian-based Linux distributions:

```
       $ apt-get install linux-headers-$(uname -r)
```

Due to the adoption of DKMS amongst major Linux distributions, many kernel modules available through package managers are (also) available as DKMS-modules. On Debian-based Linux systems, these DKMS kernel modules can be identified by their naming convention. The package names for these files end in `-dpkg`. For example: `oss4-dkms`. After installation of these packages, the kernel module source files are placed within a corresponding `/usr/src/module-version` directory together with a `dkms.conf` configuration file. Whenever a kernel or kernel module change triggers the DKMS system, the directory specified by the `source_tree` variable from `/etc/dkms/framework.conf` will be checked for the existence of subdirectories containing `dkms.conf` files. The contents of these `dkms.conf` files then determine what happens next. The following example comes from a Debian-based Linux system and should clarify this explanation:

```
$ sudo apt-get install flashcache-dkms
$ ls /usr/src/
flashcache-3.1.1+git20140801  linux-headers-3.16.0-4-amd64  linux-headers-3.16.0-4-common ↩
      linux-kbuild-3.16
$ ls /usr/src/flashcache-3.1.1+git20140801/
dkms.conf          flashcache.h        flashcache_ioctl.h  flashcache_procfs.c    ↩
    flashcache_subr.c
flashcache_conf.c  flashcache_ioctl.c  flashcache_main.c   flashcache_reclaim.c  Makefile
$ cat /usr/src/flashcache-3.1.1+git20140801/dkms.conf
BUILT_MODULE_NAME="flashcache"
DEST_MODULE_LOCATION="/kernel/drivers/block"
PACKAGE_NAME="flashcache"
PACKAGE_VERSION="3.1.1+git20140801"
AUTOINSTALL="yes"
REMAKE_INITRD="yes"
MAKE="COMMIT_REV=3.1.1+git20140801 KERNEL_TREE=$kernel_source_dir make modules"
```

`BUILT_MODULE_NAME` determines the name of the compiled module. This directive is mandatory if the DKMS-module package contains more than one module. `DEST_MODULE_LOCATION` determines the location for the compiled module. The value for this directive should always start with "/kernel" which in turn redirects to `/lib/modules/kernelversion/kernel`. This value is mandatory except for the following Linux distributions which use a distributionspecific directory: Fedora Core 6 and higher, RHEL 5 and higher, Novell SuSE Linux ES 10 and higher and Ubuntu. `PACKAGE_NAME` determines the name associated with the entire package of modules. This directive is mandatory. `PACKAGE_VERSION` determines the version associated with the entire package of modules and is mandatory. `AUTOINSTALL` is a boolean value that determines whether or not the `dkms_autoinstaller` service will try to install this module for every kernel the system boots in to. `REMAKE_INITRD` determines whether the `initrd` image should be generated again after this module is installed. The value of this directive defaults to "no". When configuring a value, know that all characters after the first character are discared. The first character is only interpreted if it is a "y" or "Y". `MAKE` is one of many directives that stores its value in to an array. The `MAKE` value determines the build options. When not defined, DKMS will try to build the module using a generic `MAKE` command.

On Debian-based Linux distributions, the directory `/usr/share/doc/dkms/examples` holds example configuration files. On Red Hat based Linux systems, example files can be found within the `/usr/share/doc/dkms` directory.

After a module is built by DKMS, it is part of an extensible framework. DKMS provides several commands to issue on the module. Covering all these commands reaches beyond the scope of this book. But apart from **man** `dkms` the **dkms** command

should be familiar. The **dkms** command makes it possible to `add` or `remove` modules to or from the source tree. Once part of the source tree, modules may be `build`. After building, a module may be installed onto the kernel it was build for using the `install` option. `uninstall` reverts this process. The `status` option prints information about added modules to standard output.

The example above uses a module from the package manager. DKMS is also capable of accepting archive formats containing binary modules, module sources or both. When adding modules to DKMS this way it is important that the archive also contains a valid `dkms.conf` file. Using the **dkms** `mktarball` command, such an archive can be created based on modules extracted from the current system. This tarball archive can then be imported to the source tree using the **dkms** `ldtarball` command.

Dracut

Just like DKMS may be configured to behave as an event-driven tool, **dracut** can behave in a similar way. Instead of compiling kernel modules, **dracut** can take care of generating a new `initramfs` image whenever there seems a need to do so.

Kernel runtime management and troubleshooting (201.3)

Candidates should be able to manage and/or query a 2.6.x, 3.x or 4.x kernel and its loadable modules. Candidates should be able to identify and correct common boot and run time issues. Candidates should understand device detection and management using udev. This objective includes troubleshooting udev rules.

Key Knowledge Areas:

- Use command-line utilities to get information about the currently running kernel and kernel modules.
- Manually load and unload kernel modules.
- Determine when modules can be unloaded.
- Determine what parameters a module accepts.
- Configure the system to load modules by names other than their file name.
- /proc filesystem
- Content of /, /boot , and /lib/modules
- Tools and utilities to analyse information about the available hardware
- udev rules

The following is a partial list of used files, terms, and utilities:

- `/lib/modules/kernel-version/modules.dep`
- module configuration files in `/etc`
- `/proc/sys/kernel/`
- **/sbin/depmod**
- **/sbin/rmmod**
- **/sbin/modinfo**
- **/bin/dmesg**
- **/sbin/lspci**
- **/usr/bin/lsdev**

- **/sbin/lsmod**
- **/sbin/modprobe**
- **/sbin/insmod**
- **/bin/uname**
- **/usr/bin/lsusb**
- `/etc/sysctl.conf`, `/etc/sysctl.d/`
- **/sbin/sysctl**
- udevmonitor
- **udevadm** monitor
- `/etc/udev`

Customise, build and install a custom kernel and kernel modules

In the paragraph What are Kernel Modules you have learned what kernel modules are. There are various utilities and configuration files associated with these modules. This paragraph will explain how you can use these to manage a running kernel and it's modules.

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. It is generally recommended to use **modprobe** instead, since **insmod** only returns very general error codes and **modprobe** is more specific about errors. Also, you will need to pass the complete filename to **insmod** , whereas **modprobe** will work with just the module name.

Frequently used options:

-s Write results to syslog instead of the terminal.

-v Set verbose mode.

Example 1.2 insmod

Suppose the kernel was compiled with modular support for a specific scsi card. To verify that this specific module exists, in this case `sym53c8xx.ko` exists, look for the file `sym53c8xx.ko` in the `/lib/modules/`*`kernel-version`*`/kernel/drivers/scsi/` directory:

```
# insmod sym53c8xx.ko
# echo $?
0
```

Modules can depend on other modules. In this example all the prerequisite modules had been loaded before, so this specific module loaded successfully. However, **insmod** does not know about prerequisites, it simply loads whatever you tell it to load. This may go wrong if there are missing prerequisites:

```
# insmod snd-ens1371.ko
insmod: error inserting 'snd-ens1371.ko': -1 Unknown symbol in module
# echo $?
1
```

The message indicates that the module requires a function or variable that is missing from the current kernel. This is usually caused by not having the prerequisite module(s) loaded or compiled into the kernel.

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 1.3 rmmod

```
# rmmod snd_ac97_codec
ERROR: Module snd_ac97_codec is in use by snd_ens1371
# echo $?
1
```

In this example the module could not be unloaded because it was in use, in this case by the `snd_ens1371` module. The solution is to remove the `snd_ens1371` module first:

```
# rmmod snd_ens1371
# rmmod snd_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. It also is more specific in it's error messages than **insmod**.

Modules can be inserted with optional *`symbol=value`* parameters such as *`irq=5`* or *`dma=3`* . Such parameters can be specified on the command line or by specifying them in the module configuration file, which will be explained later. If the module is dependent upon other modules these will be loaded first. The **modprobe** command determines prerequisite relationships between modules by reading the file `modules.dep` at the top of the module directory hierarchy, for instance `/lib/modules/2.6.31/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.) 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 in **syslog** instead of on 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 1.4 modprobe

Loading sound modules using **modprobe**.

```
# modprobe snd_ens1371
# echo $?
0
# lsmod
Module                  Size  Used by
snd_ens1371            20704  0
snd_rawmidi            22448  1 snd_ens1371
snd_seq_device          7436  1 snd_rawmidi
snd_ac97_codec        112280  1 snd_ens1371
ac97_bus                1992  1 snd_ac97_codec
snd_pcm                72016  2 snd_ens1371,snd_ac97_codec
snd_timer              21256  1 snd_pcm
snd                    62392  6 snd_ens1371,snd_rawmidi,snd_seq_device,snd_ac97_codec, ↩
    snd_pcm,snd_timer
soundcore               7952  1 snd
snd_page_alloc          9528  1 snd_pcm
```

All prerequisite modules for snd_ens1371 have been loaded. To remove both snd_ac97_codec and snd_ens1371 modules, use **modprobe -r snd_ac97_codec**.

Module configuration is handled in the file `/etc/modules.conf`. If this file does not exist the **modprobe** utility will try to read the `/etc/conf.modules` instead. The latter file is the historical name and is deprecated. It should be replaced by `/etc/modules.conf`.

Note

On some systems this file is called `/etc/modprobe.conf` or there are configuration files in a directory called `/etc/modprobe.d`.

More information on module configuration is given in the section on configuring modules.

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.

-f*format_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 1.5 modinfo

What information can be retrieved from the snd-ens1371 module:

```
# modinfo snd_ens1371
filename:       /lib/modules/2.6.31/kernel/sound/pci/snd-ens1371.ko
description:    Ensoniq/Creative AudioPCI ES1371+
license:        GPL
author:         Jaroslav Kysela <perex@perex.cz>, Thomas Sailer <sailer@ife.ee.ethz.ch>
alias:          pci:v00001102d00008938sv*sd*bc*sc*i*
alias:          pci:v00001274d00005880sv*sd*bc*sc*i*
alias:          pci:v00001274d00001371sv*sd*bc*sc*i*
depends:        snd-pcm,snd,snd-rawmidi,snd-ac97-codec
vermagic:       2.6.31-gentoo-r10 SMP mod_unload modversions
parm:           index:Index value for Ensoniq AudioPCI soundcard. (array of int)
parm:           id:ID string for Ensoniq AudioPCI soundcard. (array of charp)
parm:           enable:Enable Ensoniq AudioPCI soundcard. (array of bool)
parm:           spdif:S/PDIF output (-1 = none, 0 = auto, 1 = force). (array of int)
parm:           lineio:Line In to Rear Out (0 = auto, 1 = force). (array of int)
```

Configuring modules

You may sometimes need to control 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 to 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 `.ko` 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.(k)o: dependency1 dependency2 ...
```

Find below an example for `thermal` and `processor` modules (which are part of the ACPI layer):

```
/lib/modules/2.6.31/kernel/drivers/acpi/thermal.ko: \
    /lib/modules/2.6.31/kernel/drivers/thermal/thermal_sys.ko
/lib/modules/2.6.31/kernel/drivers/acpi/processor.ko: \
    /lib/modules/2.6.31/kernel/drivers/thermal/thermal_sys.ko
```

In this case both the `processor` and `thermal` module depend on the `thermal_sys` module.

All of the modules that are available on the system should be listed in the `modules.dep` file. They are listed with full path and filenames, so including their .(k)o extension. Those that are not depending on other modules are listed 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.

The **depmod -a** procedure is also necessary after any change in module dependencies.

The `/lib/modules/`*`kernel-version`*`/modules.dep` file contains a list of module dependencies. It is generated by the **depmod** command, which reads each module under `/lib/modules/`*`kernel-version`* to determine what symbols the module needs and which ones it exports. By default the list is written to the file `modules.dep`. The **modprobe** command in turn uses this file to determine the order in which modules are to be loaded into or unloaded from the kernel.

kmod versus kerneld

Both **kmod** and **kerneld** provide for dynamic loading of kernel-modules. A module is loaded when the kernel first needs it. For a description on modules see What are kernel modules [17].

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.

kerneld and **kmod** 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.

To enable the use of **kmod**, a kernel must be compiled with the kernel-option `CONFIG_KMOD` enabled. Because **kmod** is implemented as a kernel module the kernel-option `CONFIG_MODULES` needs to be enabled too.

Building A Custom Kernel

A summary on how to configure and compile a kernel to meet custom needs follows. You start with the configuration, normally using one of the **make** targets for configuration (**make** *`config`*, *`xconfig`*, *`menuconfig`*, *`gconfig`*, *`oldconfig`*), see the section on creating a .config file.

Once the kernel is configured it can be built with the command **make zImage/bzImage**. To build the modules, use **make modules** followed by **make modules_install**. To compile the kernel image and loadable modules in one step you can use the command **make all**. After configuring and compiling the kernel it can be installed with **make install**. Installing the kernel means installing the following files into the `/boot` directory:

- `System.map-2.6.x`
- `config-2.6.x`
- `vmlinuz-2.6.x`

The **depmod** command builds a `modules.dep` file which is needed by modprobe to determine the dependencies of the (newly) built modules.

Besides building and installing a kernel on a local machine, you can also create packages that allow you to distribute your newly built kernel to other machines. The common package formats are all available, each has its own **make** parameter:

- **make** *`rpm-pkg`*: builds both source and binary RPM packages.
- **make** *`binrpm-pkg`*: builds (only) binary kernel RPM package.
- **make** *`deb-pkg`*: builds the kernel package as a deb(ian) package.

/proc filesystem

The `/proc` filesystem is a pseudo-filesystem, meaning that its contents are directly presented by the kernel. It contains directories and other files that reflect the state of the running system. Some commands read the `/proc` filesystem to get information about the state of the system. You can extract statistical information, hardware information, network and host parameters and memory and performance information. The `/proc` filesystem also allows you to modify some parameters at runtime by writing values to files.

Contents of `/`, `/boot`, and `/lib/modules`

As an example the contents of these directories on a Debian system are presented.

Contents of /:

```
debian-601a:/$ ls -a
.    boot  home        lib32       media  proc  selinux  tmp  .ure vmlinuz
..   dev   initrd.img  lib64       mnt    root  srv      u8   usr
bin  etc   lib         lost+found  opt    sbin  sys      u9   var
```

The most important directories are /bin which contains the generic systemwide commands, /var which is the top level directory for data that will be constantly changing (logs etc.), /srv which contains data related to services the system provides (e.g. a webserver), /tmp to hold temporary files, /mnt where USB disks, tape-units and other devices that contain a filesystem can be mounted temporarily, /lib, /lib32 and /lib64 to hold modules and other libraries, /boot to hold files used during boot, /sbin under which we find special commands for the system administrator, /opt for third party packages, and /etc that contains the configuration files for your system.

The /usr directory contains a shadow hierarchy that mimics parts of the contents of the root directory. Hence, you will find a /usr/bin directory, a /usr/sbin directory, a /usr/lib directory and a /usr/etc directory. The 'secondary hierarchy' was meant to allow users to override or enhance the system, for example by allowing them to put local system administrator commands in /usr/sbin, and local system configuration data in /usr/etc. A special case are /usr/local and its subdirectories, which contain yet another shadow hierarchy that mimics parts of the contents of the root directory, this time containing very specific (local system) commands and configuration data. Hence, you will find commands in /bin, /usr/bin and in /usr/local/bin: the ones in /bin will be part of the distribution, the ones in /usr/bin may be commands that are non-mandatory or added later by the system administrator and finally the ones in /usr/local/bin are commands that are specific to the local system.

Note
The lost+found directory is used by the filesystem itself.

Contents of /boot:

```
debian-601a:/boot$ ls -a
.                       debian.bmp                  sarge.bmp
..                      debianlilo.bmp              sid.bmp
coffee.bmp              grub        System.map-2.6.32-5-amd64
config-2.6.32-5-amd64   initrd.img-2.6.32-5-amd64   vmlinuz-2.6.32-5-amd64
```

The most important are vmlinuz, System.map, initrd.img and config file. The grub directory resides here too. Also some bitmap images live here.

Contents of /lib/modules:

```
debian-601a:/lib/modules$ ls -a
.  ..  2.6.32-5-amd64
```

```
debian-601a:/lib/modules/2.6.32-5-amd64$ ls -a
.      kernel             modules.dep      modules.order    modules.symbols.bin
..     modules.alias      modules.dep.bin  modules.softdep  source
build  modules.alias.bin  modules.devname  modules.symbols   updates
```

See also the section on the FHS standard.

Tools and utilities to trace software and their system and library calls

strace

strace - trace system calls and signals

```
SYNOPSIS
      strace  [  -CdffhiqrtttTvxx  ] [ -acolumn ] [ -eexpr ] ...  [ -ofile ] [ -ppid ]
      ...  [ -sstrsize ] [ -uusername ] [ -Evar=val ] ...  [ -Evar ] ...  [ command  [
      arg ...  ] ]

      strace -c [ -eexpr ] ...  [ -Ooverhead ] [ -Ssortby ] [ command [ arg ...  ] ]
```

strace is a useful diagnostic, instructional, and debugging tool. System administrators, diagnosticians and trouble-shooters will find it invaluable for solving problems with programs for which the source is not readily available since they do not need to be recompiled in order to trace them. Students, hackers and the overly-curious will find that a great deal can be learned about a system and its system calls by tracing even ordinary programs. And programmers will find that since system calls and signals are events that happen at the user/kernel interface, a close examination of this boundary is very useful for bug isolation, sanity checking and attempting to capture race conditions.

In the simplest case **strace** runs the specified command until it exits. It intercepts and records the system calls which are called by a process and the signals which are received by a process. The name of each system call, its arguments and its return value are printed on standard error or to the file specified with the `-o` option.

By default **strace** reports the name of the system call, its arguments and the return value on standard error.

Example:

```
debian-601a:~$ strace cat /dev/null
execve("/bin/cat", ["cat", "/dev/null"], [/* 34 vars */]) = 0
brk(0)                                  = 0xc25000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcac8fb4000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=59695, ...}) = 0
mmap(NULL, 59695, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fcac8fa5000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0`\355\1\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1432968, ...}) = 0
mmap(NULL, 3541032, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0 ↩
    x7fcac8a38000
mprotect(0x7fcac8b90000, 2093056, PROT_NONE) = 0
mmap(0x7fcac8d8f000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, ↩
     0x157000) = 0x7fcac8d8f000
mmap(0x7fcac8d94000, 18472, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,  ↩
    -1, 0) = 0x7fcac8d94000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcac8fa4000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcac8fa3000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcac8fa2000
arch_prctl(ARCH_SET_FS, 0x7fcac8fa3700) = 0
mprotect(0x7fcac8d8f000, 16384, PROT_READ) = 0
mprotect(0x7fcac8fb6000, 4096, PROT_READ) = 0
munmap(0x7fcac8fa5000, 59695)           = 0
brk(0)                                  = 0xc25000
brk(0xc46000)                           = 0xc46000
open("/usr/lib/locale/locale-archive", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=1527584, ...}) = 0
mmap(NULL, 1527584, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fcac8e2d000
close(3)                                = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
open("/dev/null", O_RDONLY)             = 3
fstat(3, {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 3), ...}) = 0
read(3, "", 32768)                      = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
```

Another very useful feature of **strace** is its ability to connect to a running process using the `-p` flag. In combination with the `-f` flag you can connect to say a running daemon that seems to malfunction and gain insight in what it is actually doing.

strings

strings - print the strings of printable characters in files. **strings** can be useful for reading non-text files.

```
SYNOPSIS
      strings [-afovV] [-min-len]
             [-n min-len] [--bytes=min-len]
             [-t radix] [--radix=radix]
             [-e encoding] [--encoding=encoding]
             [-] [--all] [--print-file-name]
             [-T bfdname] [--target=bfdname]
             [--help] [--version] file...
```

For each file given, GNU **strings** prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file.

strings is mainly useful for determining the contents of non-text files, such as executables. Often used to check for names of environment variables and configurations files used by an executable.

ltrace

ltrace - a library call tracer

```
SYNOPSIS
      ltrace  [-CfhiLrStttV]  [-a  column] [-A maxelts] [-D level] [-e expr] [-l file-
      name] [-n nr] [-o filename] [-p pid] ... [-s strsize] [-u username] [-X  extern]
      [-x   extern]   ...   [--align=column]   [--debug=level]  [--demangle]  [--help]
      [--indent=nr] [--library=filename] [--output=filename] [--version] [command [arg
      ...]]
```

Similar to the **strace**, that intercepts and records system calls, **ltrace** intercepts and records the dynamic library calls which are called by the executed process and the signals which are received by that process. 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. By default, **ltrace** will start the program you specify on the command line and traces it until the program exits.

Example:

```
debian-601a:~$ ltrace cat /dev/null
__libc_start_main(0x401ad0, 2, 0x7fff5f357f38, 0x409110, 0x409100 <unfinished ...>
getpagesize()                                                    = 4096
strrchr("cat", '/')                                              = NULL
setlocale(6, "")                                                 = "en_US.utf8"
bindtextdomain("coreutils", "/usr/share/locale")                 = "/usr/share/locale"
textdomain("coreutils")                                          = "coreutils"
__cxa_atexit(0x4043d0, 0, 0, 0x736c6974756572, 0x7f4069c04ea8)   = 0
getenv("POSIXLY_CORRECT")                                        = NULL
__fxstat(1, 1, 0x7fff5f357d80)                                   = 0
open("/dev/null", 0, 02)                                         = 3
__fxstat(1, 3, 0x7fff5f357d80)                                   = 0
malloc(36863)                                                    = 0x014c4030
read(3, "", 32768)                                               = 0
free(0x014c4030)                                                 = <void>
close(3)                                                         = 0
exit(0 <unfinished ...>
__fpending(0x7f4069c03780, 0, 0x7f4069c04330, 0x7f4069c04330, 0x7f4069c04e40) = 0
fclose(0x7f4069c03780)                                           = 0
__fpending(0x7f4069c03860, 0, 0x7f4069c04df0, 0, 0x7f4069e13700)  = 0
fclose(0x7f4069c03860)                                           = 0
+++ exited (status 0) +++
```

The bootprocess

When using an `initrd`, the system goes through the following steps:

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

As moving the `initrd` from `/` to `/initrd` does not require to unmount it, process(es) that use files on the RAMdisk may kept running. All filesystems that were mounted will remain mounted during the move. However, if `/initrd` does not exist the move is not made and in that case any running processes that use files from the `initrd` image will prevent it from becoming unmounted. It will remain in memory until forced out of existence by the continuation of the bootprocess.

Also, even when the move can be made and so filesystems mounted under `initrd` remain accessible, the entries in `/proc/mounts` will not be updated. Also keep in mind that if the directory `/initrd` does not exist it is impossible to unmount the RAM disk image. The image will be forced out of existence during the rest of the bootprocess, so any filesystems mounted within it will also disappear and can not be remounted. Therefore, it is strongly recommendated to unmount all filesystems before switching from the `initrd` filesystem to the normal root filesystem, including the `/proc` filesystem.

The memory used for the `initrd` image can be reclaimed. To do this, the command **freeramdisk** must be used directly after unmounting `/initrd`.

Including `initrd` support to the kernel adds options to the boot command line:

initrd= This option loads the file that is specified as the initial RAM disk.

noinitrd This option causes the `initrd` data to be preserved, but it is not converted to a RAM disk and the normal root filesystem is mounted instead. The `initrd` data can be read from `/dev/initrd`. If read through `/dev/initrd`, the data can have any structure so it need not necessarily be a filesystem image. This option is used mainly for debugging purposes.

> **Note**
> The `/dev/initrd` is read-only and can be used only once. As soon as the last process has closed it, all memory is freed and `/dev/initrd` can no longer be accessed.

root=/dev/ram The `initrd` is mounted as root and subsequently **/linuxrc** is started. If there is no **/linuxrc** the normal boot procedure will be followed. Without using this parameter, the `initrd` would be moved or unloaded, however in this case, the root filesystem will continue to be the RAM disk. The advantage of this is that it allows the use of a compressed filesystem and it is slightly faster.

Hardware and Kernel Information

uname

uname prints system information such as machine type, network hostname, OS release, OS name, OS version and processor type of the host. The parameters are:

`-a, --all` print all information, in the order below, except omit -p and -i if these are unknown.

`-s, --kernel-name` prints the kernel name

`-n, --nodename` prints the network node hostname

-r, --kernel-release prints the kernel release

-v, --kernel-version prints the kernel build version, date and time

-m, --machine prints the machine hardware architecture name

-p, --processor prints the name of the processor type

-i, --hardware-platform prints the name of hardware platform

-o, --operating-system prints the name of the operating system

Example:

```
debian-601a:~$ uname -a
Linux debian-601a 2.6.32-5-amd64 #1 SMP Mon Mar 7 21:35:22 UTC 2011 x86_64 GNU/Linux
```

`/proc/sys/kernel/` is a directory in the `/proc` pseudo filesystem. It contains files that allow you to tune and monitor the Linux kernel. Be careful, as modification of some of the files may lead to a hanging or dysfunctional system. As these parameters are highly dependant on the kernel verions, it is advisable to read both documentation and source before actually making adjustments. See also the section on the /proc filesystem.

Some files are quit harmless and can safely be used to obtain information, for instance to show the version of the running kernel:

```
debian-601a:~$ cat /proc/sys/kernel/osrelease
```

Some files can 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:

```
debian-601a:~$ echo 20 > /proc/sys/kernel/panic
```

lspci

With **lspci** you can display information about all the PCI buses in the system and all the devices that are connected to them. Keep in mind that you need to have Linux kernel 2.1.82 or newer. With older kernels direct hardware access is only available to root. To make the output of **lspci** more verbose, one or more **-v** paramaters (up to 3) can be added. Access to some parts of the PCI configuraion space is restricted to root on many operating systems. So **lspci** features available to normal users are limited.

Example of **lspci** output on a system running Debian in Virtualbox:

```
debian-601a:~$ lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:02.0 VGA compatible controller: InnoTek Systemberatung GmbH VirtualBox Graphics ↩
    Adapter
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller (rev ↩
    02)
00:04.0 System peripheral: InnoTek Systemberatung GmbH VirtualBox Guest Service
00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio Controller ( ↩
    rev 01)
00:06.0 USB Controller: Apple Computer Inc. KeyLargo/Intrepid USB
00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:0d.0 SATA controller: Intel Corporation 82801HBM/HEM (ICH8M/ICH8M-E) SATA AHCI ↩
    Controller (rev 02)
```

lsusb

lsusb is similiar to **lspci**, but checks the USB buses and devices. To make use of **lsusb** a Linux kernel which supports the `/proc/bus/usb` inferface is needed (Linux 2.3.15 or newer). **lsusb -v** will provide verbose information.

Example of **lsusb** output as generated on a laptop running Ubuntu:

```
ubuntu:/var/log$ lsusb
Bus 007 Device 003: ID 03f0:0324 Hewlett-Packard SK-2885 keyboard
Bus 007 Device 002: ID 045e:0040 Microsoft Corp. Wheel Mouse Optical
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 002: ID 147e:2016 Upek Biometric Touchchip/Touchstrip Fingerprint Sensor
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 04f2:b018 Chicony Electronics Co., Ltd 2M UVC Webcam
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

lsdev

lsdev displays information about installed hardware. It gathers information from interupts, DMA files, and interrupts from the `/proc` directory. **lsdev** gives a quick overview about which device is using what I/O address and IRQ/DMA channels. Files being used:

```
/proc/interrupts
```

```
/proc/ioports
```

```
/proc/dma
```

Example of **lsdev** output on a Debian based system:

```
debian-601a:/var/log$ lsdev
Device            DMA   IRQ  I/O Ports
------------------------------------------------
0000:00:01.1                 0170-0177 01f0-01f7 0376-0376 03f6-03f6 d000-d00f
0000:00:03.0                 d010-d017
0000:00:04.0                 d020-d03f
0000:00:05.0                 d100-d1ff d200-d23f
0000:00:0d.0                 d240-d247 d250-d257 d260-d26f
82801AA-ICH               5
ACPI                         4000-4003 4004-4005 4008-400b 4020-4021
ahci                           d240-d247   d250-d257   d260-d26f
ata_piix              14 15    0170-0177   01f0-01f7   0376-0376   03f6-03f6   d000-d00f
cascade             4     2
dma                          0080-008f
dma1                         0000-001f
dma2                         00c0-00df
e1000                          d010-d017
eth1                     10
fpu                          00f0-00ff
i8042                  1 12
Intel                          d100-d1ff   d200-d23f
keyboard                     0060-0060 0064-0064
ohci_hcd:usb1            11
PCI                          0cf8-0cff
pic1                         0020-0021
pic2                         00a0-00a1
rtc0                      8    0070-0071
rtc_cmos                     0070-0071
timer                     0
timer0                       0040-0043
timer1                       0050-0053
vboxguest                 9
vga+                         03c0-03df
```

Note
On some systems **lsdev** is missing, use **procinfo** instead.

sysctl

sysctl is used to modify kernel paramaters at runtime. The parameters available are those listed under **/proc/sys/**. The configuration file can usually be found in `/etc/sysctl.conf`. It is important to know that modules loaded after **sysctl** is used may override its settings. You can prevent this by running **sysctl** only after all modules are loaded.

dmesg

With the **dmesg** command you can write kernel messages to standard output. Or write them to a file using **dmesg > /var/log/-boot.messages**. This can be helpful when it comes to troubleshooting issues with your system or when you just want to get some information about the hardware in your system. The output of **dmesg** can usually be found in `/var/log/dmesg`. Use **dmesg** in combination with **grep** to find specific information.

udev

Candidates should understand device detection and management using **udev**. This objective includes troubleshooting udev rules

Key Knowledge Areas:

- udev rules
- Kernel interfaces

udev was designed to make Linux device handling more flexible and safe by taking device handling out of system space into userspace.

udev consists of a userspace daemon (**udevd**) which receives "uevents" from the kernel. Communication between the userspace daemon and the kernel is done through the `sysfs` pseudo filesystem. The kernel sends the aforementioned "uevents" when it detects addition or removal of hardware, for example when you plug in your camera or USB disk. Based on a set of rules to match these uevents the kernel provides a dynamic device directory containing only device files for devices that are actually present, and can fire up scripts etc. to perform various actions. Hence, it is possible, for example, to plug in a camera, which will be automatically detected and properly mounted in the right place. After that a script might be started that copies all files to the local disk into a properly named subdirectory that will be created using the proper rights.

/etc/udev/

The `/etc/udev/` directory contains the configuration and the rule files for the udev utility. The `udev.conf` is the main configuration file for udev. Here, for instance, the logging priority can be changed.

udev rules

udev rules are read from files located in the default rules directory. `/lib/udev/rules.d/`. Custom rules to override these default rules are specified in the `/etc/udev/rules.d/` directory.

When devices are initialized or removed, the kernel sends an 'uevent'. These uevents contain information such as the subsystem (e.g. net, sub), action and attributes (e.g. mac, vendor). **udev** listens to these events, matches the uevent information to the specified rules, and responds accordingly.

A **udev** rule consists of multiple key value pairs separated by a comma. Each key value pair represents either a match key, that matches to information provided with the uevent or an assign key, which assign values, names and actions to the device nodes maintained by **udev**:

```
SUBSYSTEM=="net", ACTION=="ADD", DRIVERS="?*", ATTR{address}=="00:21:86:9e:c2:c4",
ATTR{type}=="1", KERNEL="eth*", NAME=="eth0"
```

The rule specified above would add a device `/dev/eth0` for a network card with MAC address 00:21:86:9e:c2:c4

As shown, a key value pair also specifies an operator. Depending on the used operator, different actions are taken. Valid operators are:

==	Compare for equality
!=	Compare for inequality
=	Assign a value to a key. Keys that represent a list, are reset and only this single value is assigned
+=	Add the value to a key finally; disallow any later changes, which may be used to prevent changes by any later rules

When specifying rules, be careful not to create conflicting rules (e.g. do not point two different netword cards to the same device name). Also, changing device names into something else could cause userpsace software incompatibilities. You have power, use it wisely.

udevmonitor

udevmonitor will print **udev** and kernel uevents to standard output. You can use it to analyze event timing by comparing the timestamps of the kernel uevent and the **udev** event. Usually **udevmonitor** is a symbolic link to **udevadm**. In some distributions, this symbolic link no longer exists. To access the monitor on these systems, use the command **udevadm monitor**.

Questions and answers

Linux Kernel

1. *Which technique is used in modern kernels to reduce their disk footprint?*

   Kernel images are compressed to reduce diskspace. Types of kernel images [16]

2. *What is the purpose of using kernel modules?*

   In order to only use resources when needed, kernel modules can be loaded into the kernel on demand. Also, modules can be changed and reloaded without the need to reboot the kernel. What are kernel modules [17]

3. *Why is running the command **make mrproper** generally a good idea?*

   As it prevents problems with the subsequent configuration and/or build of the kernel. Using mrproper [19]

4. *What is the most important advantage of running **make config** in stead of **make menuconfig**, **make xconfig** or **make gconfig**?*

   The command does not depend on any full-screen display capabilities and is therefore useable on slow links and systems with limited display capabilities. Why use make config? [21]

5. *What needs to be done after running **make modules**?*

   make modules_install is used after succesfully building the modules (device drivers and other items that were configured as modules). This installs them under `/lib/modules/` *kernel-version*. How to install modules? [25]

6. *For which distributions should you use **mkinitramfs** for creating an initrd image?*

   The *debian(-based)* distributions use **mkinitramfs**. Create an ignited image using mkinitramfs [48]

7. *By which means can a patch be removed from a production kernel?*

   Either apply the patch again or run the command **patch** with the **-R** parameter. Removing a kernel patch from a production kernel [29]

8. *Why is using **modprobe** recommended instead of using **insmod** to insert modules?*

   Modprobe is more specific about errors. It also works with just the module name and it can handle prerequisite relationships between modules. Using modprobe [32]

9. *By which means may the version of the running kernel be determined?*

   You could check by using**cat /proc/sys/kernel/osrelease** or by using **uname -r**. Obtain version of the running kernel [42]

10. *Which command generates the file `/lib/modules/kernel-version/modules.depfile` (a list of module dependencies)?*

    This file is generated by the **depmod** command. Obtain version of the running kernel [42]

11. *Which technique is used in modern kernels to reduce their memory footprint?*

    On demand loadable modules are used to reduce a kernel's memory footprint. Types of kernel images [16]

12. *For which reason was **udev** designed?*

    It was designed to make Linux device handling more flexible by taking device handling out of the kernel into user space. udev device handling [44]

Chapter 2

System Startup (202)

This topic has a total weight of 9 points and contains the following 3 objectives:

Objective 202.1 Customizing SysV-init system startup (3 points) Candidates should be able to query and modify the behaviour of system services at various run levels. A thorough understanding of the init structure and boot process is required. This objective includes interacting with run levels.

Objective 202.2 System recovery (4 points) Candidates should be able to properly manipulate a Linux system during both the boot process and during recovery mode. This objective includes using both the init utility and init-related kernel options. Candidates should be able to determine the cause of errors in loading and usage of bootloaders. GRUB version 2 and GRUB Legacy are the bootloaders of interest.

Objective 202.3 Alternate Bootloaders (2 points) Candidates should be aware of other bootloaders and their major features.

Customizing system startup (202.1)

Candidates should be able to query and modify the behaviour of system services at various targets / run levels. A thorough understanding of the systemd, SysV Init and the Linux boot proces is required. This objective includes interacting with systemd targets and SysV init run levels.

Key Knowledge Areas

- Systemd
- SysV init
- Linux Standard Base Specification (LSB)

Terms and Utilities

- `/usr/lib/systemd/`
- `/etc/systemd/`
- `/run/systemd/`
- **systemctl**
- **systemd-delta**
- `/etc/inittab`

- `/etc/init.d/`
- `/etc/rc.d/`
- **chkconfig**
- **update-rc.d**
- **init and telinit**

Create `initrd` using mkinitrd

Note
mkinitrd was discussed in a previous section, which also discusses how to create such an image manually.

To limit the size of the kernel, often initial ramdisk (initrd) images are used to preload any modules needed to access the root filesystem.

The **mkinitrd** is a tool which is specific to RPM based distributions (such as Red Hat, SuSE, etc.). This tool automates the process of creating an `initrd` file, thereby making sure that the relatively complex process is followed correctly.

In most of the larger Linux distributions the `initrd` image contains almost all necessary kernel modules; very few will be compiled directly into the kernel. This enables the deployment of easy fixes and patches to the kernel and its modules through RPM packages: an update of a single module will not require a recompilation or replacement of the whole kernel, but just the single module, or worst case a few dependent modules. Because these modules are contained within the `initrd` file, this file needs to be regenerated every time the kernel is (manually) recompiled, or a kernel (module) patch is installed. Generating a new `initrd` image is very simple if you use the standard tool provided on many distributions:

```
# mkinitrd initrd-image kernel-version
```

Useful options for **mkinitrd** include:

--version This option displays the version number of the **mkinitrd** utility.

-f By specifying this switch, the utility will overwrite any existing image file with the same name.

--builtin= This causes **mkinitrd** to assume the module specified was compiled into the kernel. It will not look for the module and will not show any errors if it does not exist.

--omit-lvm-modules, ***--omit-raid-modules***, ***--omit-scsi-modules*** Using these options it is possible to prevent inclusion of, respectively, LVM, RAID or SCSI modules, even if they are present, or the utility would normally include them based on the contents of `/etc/fstab` and/or `/etc/raidtab`.

Create `initrd` using mkinitramfs

Dissatisfied with the tool the RPM based distributions use (**mkinitrd**), some Debian developers wrote another utility to generate an `initrd` file. This tool is called **mkinitramfs**. The **mkinitramfs** tool is a shell script which generates a gzipped cpio image. It was designed to be much simpler (in code as well as in usage) than **mkinitrd**. The script consists of around 380 lines of code.

Configuration of mkinitramfs is done through a configuration file: `initramfs.conf`. This file is usually located in `/etc/initramfs-tools/initramfs.conf`. This configuration file is sourced by the script - it contains standard **bash** declarations. Comments are prefixed by a #. Variables are specified by:

```
variable=value
```

Options which can be used with **mkinitramfs** include:

-d confdir This option sets an alternate configuration directory.

-k "Keep" the temporary directory used for creating the image.

-o outfile Write the resulting image to `outfile`.

-r root Override the *ROOT* setting in the `initramfs.conf` file.

Note
On Debian(-based) distributions you should *always* use **mkinitramfs** as **mkinitrd** is broken there for more recent kernels.

Setting the root device

Setting the root device is one of the many kernel settings. Kernel settings originate from or can be overwritten by:

- defaults as set in the source code,
- defaults as set by the **rdev** command,
- values passed to the kernel at boot time, for example *root=/dev/xyz*
- values specified in the GRUB configuration file.

The most obvious to use are block devices like harddisks, SAN storage, CDs or DVDs. You can even have a NFS mounted root-disk, this requires usage of `initrd` and setting the *nfs_root_name* and *nfs_root_addrs* boot options. You can set or change the root device to almost anything from within the `initrd` environment. In order to do so, make sure that the `/proc` filesystem was mounted by the scripts in the `initrd` image. The following files are available in `/proc`:

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

The `real-root-dev` refers to the node number of the root file system device. It can be easily changed by writing the new number to it:

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

This will change the real root to the filesystem on `/dev/hda1`. If you wish to use an NFS-mounted root, the files `nfs-root-name` and `nfs-root-addrs` have to be set using the appropriate values and the device number should be set to 0xff:

```
# 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
```

Note
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.

The Linux Boot process

There are seven phases distinguishable during boot:

1. Kernel loader loading, setup and execution
2. Register setup

3. Kernel decompression
4. Kernel and memory initialization
5. Kernel setup
6. Enabling of remaining CPU's
7. Init process creation

The boot process is described in detail at *Gustavo Duarte's "The Kernel Boot Process"*

The kernel's final step in the boot process [1] tries to execute these commands in order, until one succeeds:

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

If none of these succeed, the kernel will panic.

The `init` process

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** allowing users to log in. It also defines "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 can be used as you wish. Most of the Unix(/Linux) variants don't use these runlevels. On a typical Debian Linux System for instance, the `/etc/rc<runlevel>.d` directories, which we will discuss later, are not available for these runlevels, though that would be perfectly legal.

runlevel s or S Runlevels s and S are internally the same. It 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 scripts to start or stop processes within runlevel "A" are executed but the actual runlevel remains "2".

[1] As defined in kernel **function kernel_init()**, formerly known as **init_post()**. Source: d6b212380... (git.kernel.org)

Configuring /etc/inittab

As mentioned before **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 "ACTION" FIELD 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' 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 **getty**s insist on doing their own housekeeping.

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 have 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 know by now this means 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.

update-rc.d

Note
This section only applies to Debian (based) distributions

Debian derived Linux distributions use the **update-rc.d** command to install and remove the init script links mentioned in the previous section.

If you have a startup script called "foobar" in `/etc/init.d/` and want to add it to the default runlevels, you can use:

```
# update-rc.d foobar defaults
Adding system startup for /etc/init.d/foobar ...
 /etc/rc0.d/K20foobar -> ../init.d/foobar
 /etc/rc1.d/K20foobar -> ../init.d/foobar
 /etc/rc6.d/K20foobar -> ../init.d/foobar
 /etc/rc2.d/S20foobar -> ../init.d/foobar
 /etc/rc3.d/S20foobar -> ../init.d/foobar
 /etc/rc4.d/S20foobar -> ../init.d/foobar
 /etc/rc5.d/S20foobar -> ../init.d/foobar
```

update-rc.d will create K (stop) links in rc0.d, rc1.d and rc6.d, and S (start) links in rc2.d, rc3.d, rc4.d and rc5.d.

If you do not want an installed package to start automatically, use **update-rc.d** to remove the startup links, for example to disable starting **dovecot** on boot:

```
# update-rc.d -f dovecot remove
Removing any system startup links for /etc/init.d/dovecot ...
 /etc/rc2.d/S24dovecot
 /etc/rc3.d/S24dovecot
 /etc/rc4.d/S24dovecot
 /etc/rc5.d/S24dovecot
```

The `-f` (force) option is required if the rc script still exists. If you install an updated **dovecot** package, the links will be restored. To prevent this create "stop" links in the startup runlevel directories:

```
# update-rc.d -f dovecot stop 24 2 3 4 5 .
Adding system startup for /etc/init.d/dovecot ...
 /etc/rc2.d/K24dovecot -> ../init.d/dovecot
 /etc/rc3.d/K24dovecot -> ../init.d/dovecot
 /etc/rc4.d/K24dovecot -> ../init.d/dovecot
 /etc/rc5.d/K24dovecot -> ../init.d/dovecot
```

Note
Don't forget the trailing . (dot).

Using systemd targets

Instead of predefined runlevels, systemd uses targets to define the system state. These targets are represented by target units. Target units end with the **.target** file extensions and their only purpose is to group together other systemd units through a chain of dependencies. This also means that, in comparison to the init runlevels, multiple targets can be active at the same time.

For example, the graphical.target unit, start services as the GNOME Display Manager but also depends on multi-user.target (which is the non-graphical system state) which in turn depends on basic.target.

Before we will continue with managing and using the targets you should know which directories are used to store the default target files and how you can override them.

As with all units the default target files are stored in **/usr/lib/systemd**, the files in this directory are created by the vendor of the software you've installed and these should never be changed except by installation scripts. The directory where you can store you custom targets and overrides is **/etc/systemd**, everything written in this directory takes precendence over the files in **/usr/lib/systemd**.

There are multiple ways to override or append properties to unit files. You can create a completely new file with the same name, for example ssh.server, in the **/etc/systemd/system**. This will override the complete unit definition. If you only want to append or change some properties you can create a new directory in **/etc/systemd/system** with the name of the unit with **.d** appended, for example sshd.server.d. In this directory you can create files with a **.conf** extension in which you can place the properties you lik to append or override. Both of these ways can also be done by using the **systemctl** command, this is further described in chaper 206.3.

There is also a third location available in which you can place files to override your unit definitions, this is the **/run/systemd** directory. Defenitions in this directory take precedence over the files in **/usr/lib/systemd** but not over those in **/etc/systemd**. Overrides in **/run/systemd** will only be used until the system is rebooted since all files in **/run/systemd** will be deleted at a reboot of the system.

To get an overview of all overrides active you can use the **systemd-delta** command. This command can return the following types:

masked Masked units, units that can't be started.

equivalent Overridden files that do not differ in content.

redirected Symbolic links to other unit files.

overridden Overridden unit files.

extended Extended unit files using a .conf file.

You can also filter by the types above using the `-t` of `--type=` flags, these take a list of the above types. If you also want to see unchanged files you can add **unchanged** as a type. Other options you can use are `--diff=false` if you don't want **systemd-delta** to show the diffs of overridden files, and `--no-pager` if you don't want the output piped to a pager.

Now that we know where the unit files are stored and how we can override them we can take a look at changing system states.

For getting and changing the (default) system state we use the **systemctl** command. A full explanation of the possibilities for this command can be found in chapter 206.3.

If we want to get the default target we can run the following command:

```
$ systemctl get-default
```

This will output the current default target which is used at boot. To get a list of all currently loaded target units you can run the following commandL

```
$ systemctl list-units --type=target
```

This will give you a list of all currently active targets. To get all available target units you can run the following command:

```
$ systemctl list-unit-files --type=target
```

If you want to change the default target you can use the **systemctl set-default** command. For example, to set the default target to multi-user.target you can run the following command:

```
$ systemctl set-default multi-user.target
```

This command will create a symbolic link **/etc/systemd/system/default.target** which links to the target file in **/usr/lib/systemd/system**.

A comparison between the init runlevels and the system targets:

You can also change the system state at runtime, this can be done using the **systemctl isolate** command. This will stop all services not defined for the chosen target, and start all the services that are defined for the target.

For example, to go to rescue mode, you can run the following command:

```
$ systemctl isolate rescue.target
```

This will stop all services except those defined for the rescue target. Note that this command will not notify any logged in users of the action.

There are also some shortcuts available to change the system state, these have an added bonus that they will notify the logged in users of the action. For example, another way to get the system into rescue mode is the following:

```
$ systemctl rescue
```

Runlevel	Target	Description
0	runlevel0.target, poweroff.target	Shutdown and poweroff the system.
1	runlevel1.target, rescue.target	Set up a rescue shell.
2	runlevel2.target, multi-user.target	Set up a non-graphical multi-user system.
3	runlevel3.target, multi-user.target	Set up a non-graphical multi-user system.
4	runlevel4.target, multi-user.target	Set up a non-graphical multi-user system.
5	runlevel5.target, graphical.target	Set up a graphical multi-user system.
6	runlevel6.target, reboot.target	Shutdown and reboot the system.

Table 2.1: Runlevels and targets

This will first notify the logged in users of the action and then stop all services not defined in the target. If you don't want the users to get notified you can add the `--no-wall` flag.

If your system is too broken to use rescue mode there is also an emergency mode available. This can be started by using the following command:

```
$ systemctl emergency
```

If you want to start certain services at boot you have to enable them. You can enable services using the **systemctl enable** command. For example, to enable sshd you run the following command:

```
$ systemctl enable sshd.service
```

To disable the service again you use **systemctl disable**. For example:

```
$ systemctl disable sshd.service
```

The unit file of the service defines at which system state the service will be running. This is configured by setting the **WantedBy=** option under the **[Install]** section. If this is set to `multi-user.target` than it will run in both non-graphical as graphical mode.

The LSB standard

The Linux Standard Base (LSB) defines an interface for application programs that are compiled and packaged for LSB-conforming implementations. Hence, a program which was compiled in an LSB compatible environment will run on any distribution that supports the LSB standard. LSB compatible programs can rely on the availability of certain standard libraries. The standard also includes a list of mandatory utilities and scripts which define an environment suitable for installation of LSB-compatible binaries.

The specification includes processor architecture specific information. This implies that the LSB is a *family* of specifications, rather than a single one. In other words: if your LSB compatible binary was compiled for an Intel based system, it will not run on, for example, an Alpha based LSB compatible system, but will install and run on any Intel based LSB compatible system. The LSB specifications therefore consist of a common and an architecture-specific part; "LSB-generic" or "generic LSB" and "LSB-arch" or "archLSB".

The LSB standard lists which generic libraries should be available, e.g. **libdl**, **libcrypt**, **libpthread** and so on, and provides a list of processor specific libraries, like **libc** and **libm**. The standard also lists searchpaths for these libraries, their names and format (ELF). Another section handles the way dynamic linking should be implemented. For each standard library a list of functions is given, and data definitions and accompanying header files are listed.

The LSB defines a list of 130+ commands that should be available on an LSB compatible system, and their calling conventions and behaviour. Some examples are **cp**, **tar**, **kill** and **gzip**, and the runtime languages **perl** and **python**.

The expected behaviour of an LSB compatible system during system initialization is part of the LSB specification. So is a definition of the **cron** system, and are actions, functions and location of the **init** scripts. Any LSB compliant init script should

be able to handle the following options: `start`, `stop`, `restart`, `force-reload` and `status`. The `reload` and `try-restart` options are optional. The standard also lists the definitions for runlevels and listings of user- and groupnames and their corresponding UID's/GID's.

Though it is possible to install an LSB compatible program without the use of a package manager (by applying a script that contains only LSB compliant commands), the LSB specification contains a description for software packages and their naming conventions.

Note
LSB employs the Red Hat Package Manager standard. Debian based LSB compatible distributions may read RPM packages by using the **alien** command.

The LSB standards frequently refers to other well known standards, for example ISO/IEC 9945-2009 (Portable OS base, very Unix like). Any LSB conforming implementation needs to provide the mandatory portions of the file system hierarchy as specified in the *Filesystem Hierarchy Standard (FHS)* , and a number of LSB specific requirements. See also the section on the FHS standard.

The bootscript environment and commands

Initially, Linux contained only a limited set of services and had a very simple boot environment. As Linux aged and the number of services in a distribution grew, the number of initscripts grew accordingly. After a while a set of standards emerged. Init scripts would routinely include some other script, which contained functions to start, stop and verify a process.

The LSB standard lists a number of functions that should be made available for runlevel scripts. These functions should be listed in files in the directory `/lib/lsb/init-functions` and need to implement (at least) the following functions:

1. **start_daemon** `[-f] [-n nicelevel] [-p pidfile] pathname [args...]`

   runs the specified program as a daemon. The **start_daemon** function will check whether the program is already running. If so, it will not start another copy of the daemon unless the `-f` option is given. The `-n` option specifies a nice level.

2. **killproc** `[-ppidfile] pathname [signal]`

   will stop the specified program, trying to terminate it using the specified signal first. If that fails, the `SIGTERM` signal will be sent. If a program has been terminated, the `pidfile` should be removed if the terminated process has not already done so.

3. **pidofproc** `[-p pidfile] pathname`

   returns one or more process identifiers for a particular daemon, as specified by the pathname. Multiple process identifiers are separated by a single space.

In some cases, these functions are provided as stand-alone commands and the scripts simply assure that the path to these scripts is set properly. Often some logging functions and function to display status lines are also included.

Changing and configuring runlevels

Changing runlevels on a running machine requires comparison of the services running in the current runlevel with those that need to be run in the new runlevel. Subsequently, it is likely that some processes need to be stopped and others need to be started.

Recall that the initscripts for a runlevel "X" are grouped in directory `/etc/rc.d/rcX.d` (or, on newer (LSB based) systems, in `/etc/init.d/rcX.d`). The filenames determine how the scripts are called: if the name starts with a "K", the script will be run with the `stop` option, if the name starts with a "S", the script will be run with the `start` option. The normal procedure during a runlevel change is to stop the superfluous processes first and then start the new ones.

The actual init scripts are located in `/etc/init.d`. The files you find in the `rcX.d` directory are symbolic links which link to these. In many cases, the start- and stop-scripts are symbolic links to the same script. This implies that such init scripts should be able to handle at least the `start` and `stop` options.

For example, the symbolic link named `S06syslog` in `/etc/init.d/rc3.d` might point to the script `/etc/init.d/syslog`, as may the symbolic link found in `/etc/init.d/rc2.d`, named `K17syslog`.

The order in which services are stopped or started can be of great importance. Some services may be started simultaneously, others need to start in a strict order. For example your network needs to be up before you can start the **httpd**. The order is determined by the names of the symbolic links. The naming conventions dictate that the names of init scripts (the ones found in the `rcN.d` directories) include two digits, just after the initial letter. They are executed in alphabetical order.

In the early days system administrators created these links by hand. Later most Linux distributors decided to provide Linux commands/scripts which allow the administrator to disable or enable certain scripts in certain runlevels and to check which systems (commands) would be started in which runlevel. These commands typically will manage both the aforementioned links and will name these in such a way that the scripts are run in the proper order.

The chkconfig command

Another tool to manage the proper linking of start up (init) scripts is **chckconfig**. On some systems (e.g. SuSE/Novell) it serves as a front-end for **insserv** and uses the LSB standardized comment block to maintain its administration. On older systems it maintains its own special comment section, that has a much simpler and less flexible syntax. This older syntax consists of two lines, one of them is a description of the service, it starts with the keyword `description:`. The other line starts with the keyword `chkconfig:`, and lists the run levels for which to start the service and the priority (which determines in what order the scripts will be run while changing runlevels). For example:

```
# Init script for foo daemon
#
# description: food, the foo daemon
# chkconfig: 2345 55 25
#
#
```

This denotes that the **foo** daemon will start in runlevels 2, 3, 4 and 5, will have priority 55 in the queue of initscripts that are run during startup and priority 25 in the queue of initscripts that are run if the daemon needs to be stopped.

The **chkconfig** utility can be used to list which services will be started in which runlevels, to add a service to or to delete it from a runlevel and to add an entire service to or to delete it from the startup scripts.

Note
We are providing some examples here, but be warned: there are various versions of **chkconfig** around. Please read the manual pages for the **chkconfig** command on your distribution first.

---

chkconfig does not automatically disable or enable a service immediately, but simply changes the symbolic links. If the **cron** daemon is running and you are on a Red Hat based system which is running in runlevel 2, the command

```
# chkconfig --levels 2345 crond off
```

would change the administration but would not stop the **cron** daemon immediately. Also note that on a Red Hat system it is possible to specify more than one runlevel, as we did in our previous example. On Novell/SuSE systems, you may use:

```
# chkconfig food 2345
```

and to change this so it only will run in runlevel 1 simply use

```
# chkconfig food 1
```

```
# chkconfig --list
```

will list the current status of services and the runlevels in which they are active. For example, the following two lines may be part of the output:

```
xdm                     0:off   1:off   2:off   3:off   4:off   5:on    6:off
xfs                     0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

They indicate that the xfs service is not started in any runlevel and the xdm service only will be started while switching to runlevel 5.

To add a new service, let's say the **foo** daemon, we create a new init script and name it after the service, in this case we might use `food`. This script is consecutively put into the `/etc/init.d` directory, after which we need to insert the proper header in that script (either the old **chkconfig** header, or the newer LSB compliant header) and then run

```
# chkconfig --add food
```

To remove the **foo** service from all runlevels, you may type:

```
# chkconfig --del food
```

Note, that the **food** script will remain in the `/etc/init.d/` directory.

System recovery (202.2)

Objectives

Candidates should be able to properly manipulate a Linux system during both the boot process and during recovery mode. This objective includes using both the init utility and init-related kernel options. Candidates should be able to determine the cause of errors in loading and usage of bootloaders. GRUB version 2 and GRUB Legacy are the bootloaders of interest.

Key Knowledge Areas

- GRUB version 2 and Legacy
- Grub shell
- Boot loader start and hand off to kernel
- Kernel loading
- Hardware initialisation and setup
- Daemon/service initialisation and setup
- Know the different boot loader install locations on a hard disk or removable device
- Overwriting standard boot loader options and using boot loader shells
- Awareness of UEFI
- UEFI and NVMe booting

Terms and Utilities

- **mount**
- **fsck**
- **inittab**, **telinit** and **init** with SysV init
- The contents of `/boot/` and `/boot/grub/`

- GRUB
- **grub-install**
- **initrd**, **initramfs**
- Master boot record

$ Resources: archNVMe,$ LPIC2sybex2nd; wikiUEFI; archUEFI; tomsUEFI; UEFI; LUtHL; Aplus901902;

GRUB explained

GRUB (short for GRand Unified Bootloader) loads the operating system kernel and transfers execution control to it.

Two major versions of GRUB exist. The current version is known as GRUB but is in fact GRUB 2. GRUB has been developed around 2011. The older version was developed back in 1999 and is now referred to as GRUB Legacy. GRUB Legacy is still in use but its development has been frozen. Unless specified otherwise, GRUB implies GRUB 2 from here on.

GRUB 2

GRUB is a modular bootloader and supports booting from PC UEFI, PC BIOS and other platforms. The advantage of its modular design is that as new filesystems and/or storage solutions are added to the kernel, boot support can easily be added to GRUB 2 in separate modules.

Examples of boot support added by such modules are modules for filesystem support (like ext4, NTFS, btrf and zfs), and LVM and software RAID devices.

GRUB is able to boot many operating systems, both free and proprietary ones. Open operating systems, like FreeBSD, NetBSD, OpenBSD, and Linux, are supported by GRUB directly. Proprietary kernels (e.g. DOS, Windows and OS/2) are supported using GRUB's chain-loading function. Chain-loading implies that GRUB will be used to boot the system, and in turn will load and run the proprietary systems bootloader, which then boots the operating system.

The GRUB boot process features both a menu interface and a command-line interface (CLI). The CLI called is called the GRUB shell and allows you to execute commands to select a root device (**root** command), load a kernel from it (**linux** command) and, if necessary load some additional kernel modules (**insmod**) and subsequently boot the kernel (**boot** command). The menu interface offers a quick selection method of the desired runtime environment. While booting, both interfaces are available. On boot the menu is displayed, and the user can simply choose one of the menu entries. Without user interaction, the system will boot the default entry after a pre-defined time value has passed.

Alternatively, the user can hit **e** to edit the current entry before booting, or hit **c** to enter the CLI. Some Linux distributions hide the GRUB screen during boot. Pressing the **SHIFT** key right after BIOS/UEFI initialization will unhide the GRUB screen.

After invoking the GRUB shell, the user can type commands from the list below. The list of commands may vary, and depends on which modules are present on the system. The **help** command will produce a list of available commands.

```
o acpi:                 Load ACPI tables
o badram:               Filter out bad regions of RAM
o blocklist:            Print a block list
o boot:                 Start up your operating system
o cat:                  Show the contents of a file
o chainloader:          Chain-load another boot loader
o cmp:                  Compare two files
o configfile:           Load a configuration file
o cpuid:                Check for CPU features
o crc:                  Calculate CRC32 checksums
o date:                 Display or set current date and time
o drivemap:             Map a drive to another
o echo:                 Display a line of text
o export:               Export an environment variable
o false:                Do nothing, unsuccessfully
o gettext:              Translate a string
```

```
  o gptsync:              Fill an MBR based on GPT entries
  o halt:                 Shut down your computer
  o help:                 Show help messages
  o initrd:               Load a Linux initrd
  o initrd16:             Load a Linux initrd (16-bit mode)
  o insmod:               Insert a module
  o keystatus:            Check key modifier status
  o linux:                Load a Linux kernel
  o linux16:              Load a Linux kernel (16-bit mode)
  o list_env:             List variables in environment block
  o load_env:             Load variables from environment block
  o loopback:             Make a device from a filesystem image
  o ls:                   List devices or files
  o normal:               Enter normal mode
  o normal_exit:          Exit from normal mode
  o parttool:             Modify partition table entries
  o password:             Set a clear-text password
  o password_pbkdf2:      Set a hashed password
  o play:                 Play a tune
  o pxe_unload:           Unload the PXE environment
  o read:                 Read user input
  o reboot:               Reboot your computer
  o save_env:             Save variables to environment block
  o search:               Search devices by file, label, or UUID
  o sendkey:              Emulate keystrokes
  o set:                  Set an environment variable
  o true:                 Do nothing, successfully
  o unset:                Unset an environment variable
  o uppermem:             Set the upper memory size
```

GRUB uses its own syntax to describe hard disks. Device names need to be enclosed in brackets, e.g

```
(fd0)
```

denotes the floppy disk, and

```
(hd0,1)
```

denotes the first partition on the first hard disk. Note that while disk numbers start at zero, partition numbers start at one, so the last example references the *first* disk and the *first* partition.

GRUB uses the computer BIOS to find out which hard drives are available. But it can not always figure out the relation between Linux device filenames and the BIOS drives. The special file `/boot/grub/device.map` can be created to map these, e.g.:

```
(fd0)  /dev/fd0
(hd0)  /dev/hda
```

Note that when you are using software RAID-1 (mirroring), you need to set up GRUB on both disks. Upon boot, the system will not be able to use the software RAID system yet, so booting can only be done from one disk. If you only set up GRUB on the first disk and that disk would be damaged, the system would not be able to boot.

GRUB Configuration File

The configuration file for GRUB 2 is `/boot/grub/grub.cfg`. The GRUB configuration file is written in a shell-like scripting language with conditional statements and functions.

It is not recommended to modify `grub.cfg` directly; the configuration file is updated whenever a kernel is added, updated, or removed using the package manager of the distribution or when the user runs the **update-grub** script. The **update-grub** is a wrapper around **grub-mkconfig**, specifying `grub.cfg` as its output file. The behaviour of **grub-mkconfig** is controlled by files in the directory `/etc/grub.d` and keywords in the `/etc/default/grub` file.

Examples keywords: the default menu entry to boot (GRUB_DEFAULT) or the timeout in seconds to boot the default menu entry after the menu is displayed (GRUB_TIMEOUT).

Operating systems, including foreign operating systems like Windows are automatically detected by the `/etc/grub.d/30_os_prober` script. A custom file (by default `40_custom`) can be modified by the user to create custom entries.

GRUB 2 menu entries start with the **menuentry** keyword. The menu entry's title can be found within quotation marks on the **menuentry** line. The **menuentry** line ends with an opening curly brace ({). The menu entry ends with a closing curly brace (}).

A very simple example:

```
menuentry 'Linux 3.3.10' {
<... >
}
```

Differences with GRUB Legacy

At first glance, the two versions do not differ much. However, there are some obvious differences:

- The GRUB configuration file is now called `/boot/grub/menu.list`, while Red Hat based distributions favor the `/boot/grub/grub.conf` filename. Besides the slightly different name, the configuration file also has a different syntax. The grub.cfg file is now generated during grub-install, and is not supposed to be edited by hand.
- The core GRUB engine is smaller and less platform dependent. Support for many different filesystems and platforms is now available in separate modules. As a consequence, the platform, and filesystem(s) in use determine the modules loaded during the boot sequence. In contrast, GRUB Legacy has a fixed boot sequence with critical components hardcoded, making it less flexible.
- Partition numbering starts at 1 in GRUB 2, rather than 0. Disks are still numbered from 0. This can be a bit confusing.
- GRUB 2 kernel specification is done with the **linux** command, while in GRUB Legacy, we use the **kernel** command instead.
- The root device can be selected with **set root** in stead of the **root** command. The root device can also be set from the **search** command which can find devices by disk label or UUID.
- GRUB 2 uses **insmod** to load modules. In GRUB Legacy modules are loaded with **module** or **modulenounzip**.

GRUB Legacy

The GRUB Legacy definitions for the menu-entries are stored in `/boot/grub/menu.lst`. On some systems you may find a `grub.conf` [2] link in the `/etc` or `/boot/grub` directory. Because GRUB accesses the file directly, any changes in that file will impact the bootloader immediately.

On systems with the Legacy bootloader, GRUB shell is available to install and emulate it. This shell emulates the boot loader and can be used to install the boot loader. It also comes in handy to inspect your current set up and modify it. To start it up (as `root`) simply type **grub**. In the following example we display the help screen:

```
# grub
grub> help
blocklist FILE                          boot
cat FILE                                chainloader [--force] FILE
color NORMAL [HIGHLIGHT]                configfile FILE
device DRIVE DEVICE                     displayapm
displaymem                              find FILENAME
geometry DRIVE [CYLINDER HEAD SECTOR [ halt [--no-apm]
help [--all] [PATTERN ...]              hide PARTITION
initrd FILE [ARG ...]                   kernel [--no-mem-option] [--type=TYPE]
makeactive                              map TO_DRIVE FROM_DRIVE
```

[2] Not to be confused with GRUB 2 `grub.cfg` config file.

```
md5crypt                                module FILE [ARG ...]
modulenounzip FILE [ARG ...]            pager [FLAG]
partnew PART TYPE START LEN             parttype PART TYPE
quit                                    reboot
root [DEVICE [HDBIAS]]                  rootnoverify [DEVICE [HDBIAS]]
serial [--unit=UNIT] [--port=PORT] [-- setkey [TO_KEY FROM_KEY]
setup [--prefix=DIR] [--stage2=STAGE2_ terminal [--dumb] [--timeout=SECS] [--
testvbe MODE                            unhide PARTITION
uppermem KBYTES                         vbeprobe [MODE]

grub >_
```

Note
Note that the grub shell is not available for GRUB 2. Instead, you can install the Grub Emulator, **grub-emu**.

Other GRUB Legacy commands include the **blocklist** command, which can be used to find out on which disk blocks a file is stored, or the **geometry** command, which can be used to find out the disk geometry. You can create new (primary) partitions using the **partnew** command, load an **initrd** image using the **initrd** command, and many more. All options are described in the GRUB documentation. GRUB is part of the GNU software library and as such is documented using the **info** system. On most systems there is a limited **man** page available as well.

The initial boot process , upon boot, the BIOS accesses the initial sector of the hard disk, the so-called MBR (Master Boot Record), loads the data found there in memory and transfers execution to it. If GRUB is used, the MBR contains a copy of the first stage of GRUB, which tries to load stage 2.

To be able to load stage 2, GRUB needs to have access to code to handle the filesystem(s). There are many filesystem types and the code to handle them will not fit within the 512 byte MBR, even less so since the MBR also contains the partitioning table. The GRUB parts that deal with filesystems are therefore stored in the so-called DOS compatibility region. That region consists of sectors on the same cylinder where the MBR resides (cylinder 0). In the old days, when disks were adressed using the CHS (Cylinder/Head/Sector) specification, the MBR typically would load DOS. DOS requires that its image is on the same cylinder. Therefore, by tradition, the first cylinder on a disk is reserved and it is this space that GRUB uses to store the filesystem code. That section is referred to as stage 1.5. Stage 1.5 is commonly referred to as the `core.img`; it is constructed from several files by the installer, based on the filesystem(s) grub needs to support during boot.

Stage 2 contains most of the boot-logic. It presents a menu to the end-user and an additional command prompt, where the user can manually specify boot-parameters. GRUB is typically configured to automatically load a particular kernel after a timeout period. Once the end-user made his/her selection, GRUB loads the selected kernel into memory and passes control on to the kernel. At this stage GRUB can pass control of the boot process to another loader using chain loading if required by the operating system.

In Linux, the **grub-install** command is used to install stage 1 to either the MBR or within a partition.

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 the GRUB prompt. What can be influenced will be discussed in the following sections, but first we must activate the prompt.

Choosing another kernel

If you have just compiled a new kernel and you are experiencing difficulties with the new kernel, chances are that you would like to revert to the old kernel.

For GRUB, once you see the boot screen, use the cursor keys to select the kernel you would like to boot, and press **Enter** to boot it.

Booting into single user mode or a specific runlevel

This can be useful if, for instance, you have installed a graphical environment which is not functioning properly. You either do not see anything at all or the system does not 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 is not running will give you access to the system so you can correct the problem.

To boot into single user mode in GRUB, point the cursor to the kernel entry you would like to boot and press **e**. Then select the line starting with "linux" (for GRUB 2) or "kernel" in GRUB Legacy. Go to the end of the line, and add "single". After that, press **Enter** to exit the editing mode and then press CTRL-x (GRUB 2), or **b** in GRUB Legacy to exit the editor and boot that entry.

Switching runlevels

It is possible in Linux to switch to a different runlevel than the currently active one. This is done through the **telinit** command. It's syntax is simple: telinit [OPTION] RUNLEVEL where RUNLEVEL is the number of the runlevel.

The only option which **telinit** supports is `-e KEY=VALUE`. It is used to specify an additional environment variable to be included in the event along with RUNLEVEL and PREVLEVEL. Usually you will not use this option.

You will find you use **telinit** mostly to switch to single-user mode (runlevel 1), for example to be able to umount a filesystem and **fsck** it. In that case you can use:

```
# telinit 1
```

Note that **telinit** on most systems is a symbolic link to the **init** command.

Use of the command **/sbin/init q** forces init to reload `/etc/inittab`.

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. This is only applicable if support for the device has been compiled into the kernel, not if you are using a loadable module.

As an example, let us pretend we have 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. For GRUB, you can add the additions the same way as booting into single-user mode, replacing the keyword "single" by the parameters you need pass.

For the example above, the keywords to pass to the kernel would be:

```
ether=5,0x300,eth0 ether=11,0x340,eth1
```

The Rescue Boot process

When fsck is started but fails

During boot file systems are checked. On a 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 wil get a message:

```
fsck failed. Please repair manually

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

If you do not press Ctrl-D but enter the root password, you will 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 (see next section) you can boot from boot media.

If your root (/) filesystem is corrupt

Using the distribution's bootmedia

A lot of distributions come with one or more CD's or boot images which can be put on a USB stick. One of these CD's usually contains a "rescue" option to boot Linux in core. This allows you to fix things.

Remember to set the boot-order in the BIOS to boot from CD-ROM or USB stick first and then HDD. In the case of a USB stick it may also be necessary to enable "USB Legacy Support" in the bios.

What the rescue mode entails is distribution specific. But it should allow you to open a shell with root-privileges. There you can run **fsck** on the unmounted corrupt filesystem.

Let's assume your root partition was `/dev/sda2`. You can then run a filesystem check on the root filesystem by typing **fsck -y /dev/sda2**. 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 the root (/) filesystem of a rescue image 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 corrected the errors, do not forget to **umount** the filesystems you have mounted before you reboot the system, otherwise you will get a message during boot that one or more filesystems have not been cleanly umounted and **fsck** will try to fix it again.

UEFI and NVMe boot considerations

For many decades, the system BIOS *(Basic Input Output System)* took care of hardware and software initialization during the boot process. Early BIOS versions required manual configuration of physical jumpers on the motherboard. Later versions replaced the manual jumper routine by a software menu, capable of providing an interface to configure the most elementary computer settings. As convenient as this may sound, the constant evolution of computer systems evolved to a point where even the most sophisticated BIOS software proved to have its limitations. To combat these limitations, Intel developed the EFI *(Extensible Firmware Interface)* system in 1998 as a BIOS replacement. The EFI system dit not catch on, until the standard was adopted by the UEFI Forum around 2005. The standard was then (re)branded from EFI to UEFI *(Universal Extensible Firmware Interface)*. UEFI is sometimes also referred to as (U)EFI. Linux kernel 3.15 and newer should be able to use the UEFI advantages.

What are these advantages you may ask? To answer that question we have to look at the BIOS limitations first. One of the limitations of BIOS systems is noticable when booting operating systems. Traditionally, a BIOS can be configured to use one or more boot devices in a specific order. A boot device can be an optical drive, a harddrive, a portable USB volume or a network interface card. After the BIOS has performed the POST *(Power On Self Test)*, each configured boot device will be checked for the existence of a boot loader. The first bootloader detected will be loaded. In case of a harddrive, the BIOS expects the bootloader

to be located at sector 0 or the MBR*(Master Boot Record)*. Since the MBR only allows for a small amount of data (446 bytes) to be stored, the MBR usually contains instructions that point to another piece of code on disk. This two stage approach is known as *chainloading*. This other piece of code could then consist of a boot manager. A boot manager is capable of loading operating systems located at various locations on the storage volumes. Both the first and second stage of the boot code have to be stored within the first MegaByte of available storage on the harddrive.

UEFI uses a different approach. Instead of being limited to the MBR contents of one specific drive, UEFI reads boot data from an ESP partition. ESP stands for *EFI System Partition*. The ESP is a designated boot partition. The filesystem is usually of the type FAT , and it can hold any size of bootloader, or even multiple ones. On Linux systems, the ESP is usually mounted as `/boot/efi`. Underneath that mountpoint will be a directory structure that depends on the Operating System in use. The boot files located within those directories carry a `.efi` extension. With UEFI, the UEFI software acts as a mini-bootloader looking for filenames ending in `.efi` within pre-defined locations. On a Fedora based system, the contents of the ESP may look as follows:

```
# cd /boot/efi/
# ls -a
.  ..  EFI
#  cd EFI
#  ls
BOOT  fedora
# ls -l BOOT
total 1332
-rw-r--r-- 1 root root 1293304 May 17  2016 BOOTX64.EFI
-rw-r--r-- 1 root root   66072 May 17  2016 fallback.efi
# ls -l fedora/
total 3852
-rw-r--r-- 1 root root     104 May 17  2016 BOOT.CSV
drwxr-xr-x 2 root root    4096 Sep 28 22:17 fw
-rwxr-xr-x 1 root root   70864 Sep 28 22:17 fwupx64.efi
-rw-r--r-- 1 root root 1276192 May 17  2016 MokManager.efi
-rw-r--r-- 1 root root 1293304 May 17  2016 shim.efi
-rw-r--r-- 1 root root 1287000 May 17  2016 shim-fedora.efi
```

In the example above, every file ending in `.efi` can add functionality to the UEFI system. So, whereas BIOS based systems depend on harddrive metadata to boot up a system, UEFI based systems are capable of reading files within the ESP portion of the harddrive. UEFI offers backwards compatibility towards legacy BIOS functions, while at the same time offering more advanced functions for modern computers. Computers using BIOS software have trouble dealing with todays 8TB harddrives. UEFI based computers are able to use GPT disk layouts that defeat the 2TB partition limit of their BIOS counterparts. The UEFI software comes with network support for IPv4 and IPv6. TCP and UDP are supported, and booting remote boot media is supported using TFTP and even HTTP. Booting over HTTP does require UEFI 2.5 or newer. Version 2.5 was released in Januari 2016.

LPIC-2 exam candidates should be aware of the possibility to switch between UEFI and Legacy BIOS boot modes on modern computers. Despite the advantages that UEFI may have, there are also requirements that should be met. The `.efi` boot files are expected to be located beneath a certain path. When *Secure Boot* is enabled, the boot code has to be digitally signed. Otherwise, systems may encounter boot issues. When troubleshooting boot issues on a modern Linux computer, try to distinguish MBR from GPT disk layouts. When using the UEFI boot mode, confirm that the Linux distribution in use can also handle UEFI boot. When *Secure Boot* is enabled, confirm that the required conditions are met. When in doubt, switch back to "Legacy BIOS" or equivalent within the UEFI interface. When booting from USB, it may be necessary to enable 'Legacy USB' settings for Mass Storage Devices in the UEFI interface.

NVMe In the previous chapter, 8TB harddrives are mentioned as a result of recent computer storage evolution. These conventional SATA *(Serial Advanced Technology Attachmenti)* harddrives have moving parts, and are controlled using a protocol called AHCI *(Advanced Host Configuration Interface)*. In recent years, SSD *(Solid State Disk)* harddrives have become more popular. One of the advantages of these drives is the lack of moving parts. This makes SSD harddrives not only more energy efficient but also faster than mechanical harddrives. Because the SSD drives have to be compatible with existing computers, they are connected with the same SATA connector mechanical harddrives use. And they also use the same AHCI protocol. This protocol was initially designed with mechanical harddrives in mind. AHCI uses 1 queue with 32 commands to control the harddrive. This poses a bottleneck for the newer generation of SSD harddrives. To combat this bottleneck, a new technology called NVMe *(Non Volatile Memory Express)* has been developed. NVMe allows SSD harddrives to connect to a NVMe controller that is connected to the PCI-E bus on the motherboard. The SSD harddisk is then controlled using the NVMHCI *(Non Volatile Memory Host*

Configuration Interface) protocol. Instead of 1 queue holding 32 commands at a time, the SSD can now be controlled using 65.000 queues holding up to 65.000 commands each. This is possible because the PCI-E bus is much faster than the SATA bus. The latest generation of fast SSD harddrives can achieve throughput speeds up to seven times faster using NVMe when compared to PCI-E connected AHCI harddrives.

Just as traditional harddrives connected to a Linux computer are represented by `/dev/hda*` or `/dev/sda*` references, NVMe harddrives are represented by `/dev/nvme*` within the Linux filesystem tree. When working with these harddrives, be aware that the disk notation starts at `0`, but the namespace and partition on disk start at `1`. Therefore, the first partition on the first namespace on the first NVMe harddrive of a system is represented by `/dev/nvme0n1p1`. More about UEFI and NVMe booting at 204.2

Alternate Bootloaders (202.3)

Candidates should be aware of other bootloaders and their major features.

Key Knowledge Areas

- SYSLINUX, ISOLINUX, PXELINUX
- Understanding of PXE for both BIOS and UEFI
- Awareness of systemd-boot and U-Boot
- Booting UEFI systems
- Systemd-boot
- U-Boot

Terms and Utilities

- `syslinux`
- `extlinux`
- `isolinux.bin`
- `isolinux.cfg`
- `isohdpfx.bin`
- `efiboot.img`
- `pxelinux.0`
- `pxelinux.cfg/`
- `uefi/shim.efi`
- `uefi/grubx64.efi`
- `UEFI`
- `Systemd-boot`
- `U-boot`

LILO

The lilo bootloader consists of two stages. During the two stage boot, LILO indicates its progress by printing consecutive letters from the word "LILO" to the BIOS console, with one letter at the start and end of each stage.

Errors during either stage result in only parts of the word being printed, optionally followed by a numeric BIOS error code, or a single character. E.g, **LIL-** or **LIL?**.

In interactive mode (**prompt** keyword in `/etc/lilo.conf`) LILO presents the user with a choice of up to 16 entries. If no **timeout=nnn** (**nnn** in tens of a second) is specified, the bootloader will wait indefinitely. If the timeout expires, or no **prompt** was included, LILO proceeds by loading the first image listed. This can be overruled with the **default=name** keyword.

/etc/lilo.conf and /sbin/lilo

Unless modern bootloaders, neither stage of LILO is actually aware of the contents of `/etc/lilo.conf`. Instead, the file is used only as a specification for the lilo installer, typically at `/sbin/lilo`. Without additional options, `/sbin/lilo` will install the bootloader into the MBR, and process the contents of `/etc/lilo.conf`, creating a mapping of any files specified to store into `/boot/map`.

Please refer to the **lilo.conf(5)** man page online for a detailed description of `/etc/lilo.conf`.

Example `/etc/lilo.conf` (from an older debian distribution)

```
# lilo.conf
#
#  global options:
boot=/dev/hda
prompt
timeout=150
lba32
compact
vga=normal
root=/dev/hda1
read-only
menu-title=" John's Computer "
#
#  bootable kernel images:
image=/boot/zImage-1.5.99
     label=try
image=/boot/zImage-1.0.9
     label=1.0.9
image=/tamu/vmlinuz
     label=tamu
     initrd=initramdisk.img
     root=/dev/hdb2
     vga=ask
#
#  other operating systems:
other=/dev/hda3
     label=dos
     boot-as=0x80    # must be C:
other=/dev/hdb1
     label=Win98
     boot-as=0x80    # must be C:
other=/dev/hdb5
     label=os2
     loader=os2_d
     table=E:   # os2 sees as E:
```

SYSLINUX, ISOLINUX, PXELINUX: The Syslinux Project

SYSLINUX is a linux bootloader designed to run from an MS-DOS/Windows FAT file system. It is limited to Intel/AMD hardware.

Over time, the Syslinux project (http://syslinux.org) expanded to include support for booting natively from CD-ROMS (**ISOLINUX**), linux file systems (**EXTLINUX**) and over PXE (**PXELINUX**).

This summary handles the LPIC-2 specific objectives. A full description can be found in the syslinux wiki at http://www.syslinux.org/-wiki/index.php/SYSLINUX

SYSLINUX

SYSLINUX is an Intel Linux bootloader which is able to boot from Windows/MS-DOS FAT based file systems. It can be installed with the command of the same name, from either Windows, MS-DOS, or linux.

The syslinux system consists of the actual bootloader and its installer. The bootloader itself is a 32bit native binary written in assembler.

The **syslinux** installer command comes in different versions: **syslinux.exe** for Windows, **syslinux** for linux. There is even a **syslinux.com** for DOS based systems.

SYSLINUX Installer Options

```
--offset        -t      Offset of the file system on the device
--directory     -d      Directory for installation target
--install       -i      Install over the current bootsector
--update        -U      Update a previous installation
--sectors=#     -S      Force the number of sectors per track
--heads=#       -H      Force number of heads
--stupid        -s      Slow, safe and stupid mode
--raid          -r      Fall back to the next device on boot failure
--once=...              Execute a command once upon boot
--clear-once    -O      Clear the boot-once command
--reset-adv             Reset auxilliary data
--menu-save=    -M      Set the label to select as default on the next boot
--force         -f      Ignore precautions
```

MSDOS specific options

```
-m      MBR: install a bootable MBR sector to the beginning of the drive.
-a      Active: marks the partition used active (=bootable)
```

Linux only

```
-t      (-o on older systems) Specifies the byte offset of the filesystem
        image in the file. It has to be used with a disk image file.
```

Exceptions for ISOLINUX, EXTLINUX, PXELINUX

The ISOLINUX Installer While **syslinux** expects a target device to write the bootloader, the **isolinux** installer generates an ISO image from a directory structure. The directory must include a subdirectory `isolinux` which in turn must include the actual `isolinux.bin` bootloader.

The EXTLINUX Installer The **extlinux** installer expects a mounted file system to install the bootloader into.

PXELINUX is handled in Section 2.3.6.

Syslinux Boot Configuration

The bootloaders installed by these utilities will look for a `syslinux.cfg` file in the following three directories: `/boot/syslinux`, `/syslinux`, `/` (root).

The ISOLINUX bootloader will first look for `/boot/isolinux` and `/isolinux`. The EXTLINUX bootloader looks for `/boot/extlinux` and `/extlinux` first.

The directory where the config file is found will be the default directory for further pathnames in the boot process.

The CONFIG keyword will restart the boot process with a new config file. If two pathnames are supplied, the second parameter overrides the default directory.

The boot process will look in the `syslinux.cfg` file for a line with "LABEL linux". When found, it will use any subsequent keywords to guide the boot process. (A **DEFAULT label** phrase can be used to override the "linux" label.)

Typical keywords in a boot configuration:

KERNEL *image* The **KERNEL**keyword specifies an image file. This does not have to be an actual kernel image, but can be the name of the next stage bootprogram. SYSLINUX e.a. rely on filename extensions to decide on the file format.

.0 is used with PXELINUX for the PXE NBP (Network Boot Program), with **pxelinux.0** being the default.

.bin used with ISOLINUX, and refers to the CD Boot Sector,

.bs or .bss refer to (patched) DOS bootsectors [SYSLINUX].

.com, .cbt, and .c32 are COMBOOT images (DOS,non-DOS,32 bit). For versions 5.00 and later **c32** changed from COMBOOT to ELF binary format.

.img is an ISOLINUX diskimage.

Any *other* file extension (or none at all) indicate a linux kernel image.

The file type can also be forced by using one of the **KERNEL** keyword aliases: **LINUX**, **BOOT**, **BSS**, **PXE**, **FDIMAGE**, **COMBOOT** or **COM32**.

APPEND string The **APPEND** keyword specifies a string of boot parameters that is appended to the kernel command line. Only the last **APPEND** line will be applied.

SYSAPPEND The **SYSAPPEND** keyword expects a numeric argument that is interpreted as a bitmap. Each bit in this bitmap will add a specific auto-generated string to the kernel command line.

Example 2.1 SYSAPPEND Examples

1:Adds a string with network information: **ip=client:bootserver:gw:netmask**
2:Adds **BOOTIF=** ..., identifying the active network interface by its mac address.
4:Adds the string **SYSUUID=**...
8:Add **CPU=**...
Higher order bits (0x00010 through 0x10000) control additional strings from DMI/SMBIOS, if available. A full list can be found in the Syslinux wiki: http://www.syslinux.org/wiki/index.php/SYSLINUX"

INITRD *filename* The **INITRD** keyword is equivalent to **APPEND initrd=filename**.

TIMEOUT num For interactive use, the argument to **TIMEOUT** indicates the number of tens of a second that SYSLINUX should wait for input on the console or serial port.

PXELINUX

The PXELINUX bootloader is used as the second stage of a PXE network boot. The PXE network boot mechanism is further explained in Section 2.3.7.

PXELINUX expects a standard TFTP server with a `/tftpboot` directory containing the `pxelinux.0` syslinux bootloader, and the `ldlinux.c32` library module.

In addition, a directory `/tftpboot/pxelinux.cfg` must exist for additional configuration details.

A PXE TFTP boot server can serve many different clients, and needs a way to maintain different configuration files for different (categories of) clients. There are many different ways in which the name of the configuration file can be specified.

Combine DHCP Option 209 and 210 Option 209 (**pxelinux.config-file**) specifies the filename for the configfile.

Option 210 (**pxelinux.pathprefix**) specifies the search path (directory prefix) on the TFTP server namespace (ending in the OS-specific separator character for the file system).

Hardcoded in the pxelinux.0 image. The **pxelinux-options** command can be used to hardcode the options as shown in Figure 2.1.

```
  6     (domain-name-servers),
 15     (domain-name),
 54     (next-server),
209     (config-file),
210     (path-prefix),
211     (reboottime).
```

Figure 2.1: pxelinux.0 embedded options (optional)

Options can be specified as 'before-options', where DHCP has precedence, or as 'after-options', which override DHCP.

Derived from UUID, MAC-address, or IP-Address If no config file is specified, the filename is derived from a list of variables. The first file in the list that actually exists on the TFTP server will be used.

The list of variables is:

- The client's UUID, in lower case.
- The client's MAC address, in lower case hexadecimal, with bytes separated by a dash ("-").
- The longest possible prefix of the Upper case hexadecimal representation of the client's ipv4 address. Each time the string does not match, PXELINUX drops the last character from the string and tries again as long as the result contains at least one character.
- As a last resort, PXELINUX will try to retrieve the file named "default".

Understanding PXE

PXE is a specification created by Intel to enhance the original network boot protocols: BOOTP, TFTP and DHCP.

BOOTP, RARP and TFTP were created by the IETF to enable systems to automatically retrieve their network configuration and initial bootloader from a server.

The initial BOOTP standard was limited to a number of fixed fields in which client and server could exchange information. A client could supply its hardware address in **chaddr**, and request a specific type of **file**, and would receive its ip address as **yiaddr** and a servername **sname**. Combined with the server IP address field (**siaddr**) and the gateway IP address field, and the returned boot file name (**file**) this would tell the boot client where to retrieve its boot image, using TFTP.

BOOTP Fields

```
ciaddr  4       client IP address
yiaddr  4       your IP address
siaddr  4       server IP address
giaddr  4       gateway IP address
chaddr  16      client hardware address
sname   64      optional server host name, null terminated string.
file    128     boot file name, null terminated string;
vend    n     n=64 in original BOOTP, starts with the 4 byte
                DHCP 'magic' number.
```

Over time networks and IT infrastructure became more complicated and requirements more demanding. To allow clients to provide more information about themselves and to retrieve tailored information, BOOTP received the BOOTP Vendor Information Extensions [RFC 1048], which in turn was enhanced with a new protocol, DHCP. DHCP extended BOOTP with a number of standard options, defining different types of messages. Some DHCP options may overlap with standard BOOTP fields, and should contain the same value in that case.

Note
A DHCP message is a BOOTP packet (request or response) with a special 4 byte value (the DHCP magic cookie) in the BOOTP "Vendor Information Field". Following that are DHCP options, consisting of a single byte option type, a length field, and **length** bytes of option content.
This rule has two exceptions: *Padding* (**0**) and *End of Options* (**255**) are just one byte in length and lack a length field.

Finally, Intel introduced PXE, to enhance the BOOTP/DHCP protocol even further, in an attempt to standardise the way clients can identify themselves. This allows boot clients and servers to minimize the number of packets that needs to be exchanged before they can decide on the correct parameters and the boot program needed to get going.

A PXE boot request starts with a DHCP Discover message including at least five options, of which three are PXE-specific:

```
(53) DHCP Message type (DHCP Discover),
(55) Parameter Request List,
(93) Client System Architecture,
(94) Client Network Device Interface,
(97) UUID/GUID-based Client Identifier
```

Options 93, 94, and 97 are defined in the PXE specification. In addition, option 55, the Parameter Request List, must *also* request options 128 through 135, even though a server is not required to provide a response to them. This list and the three options listed above act to identify the client as PXE aware.

Proxy DHCP for PXE

Not every DHCP server (especially those embedded in network equipment) will be able to process a PXE request.

The PXE specification allows PXE-aware DHCP servers to co-exist with simple DHCP servers, where the default DHCP server provides the basic network detail. The PXE-aware server can then provide additional detail for the actual TFTP boot process. This is called proxy-DHCP.

It is even possible to separate DHCP services on the same server, in which the proxy DHCP service is expected to listen to UDP port 4011.

Example DHCP request

See below for an example DHCP Discover message, including requests for standard network detail such as (1) Subnet Mask, (3) Router, (6) Name server, (12) Host Name, (15) Domain Name, etc.

Example 2.2 DHCP Discover message

```
Option: (53) DHCP Message Type
    Length: 1
    DHCP: Discover (1)
Option: (57) Maximum DHCP Message Size
    Length: 2
    Maximum DHCP Message Size: 1464
Option: (55) Parameter Request List
    Length: 35
    Parameter Request List Item: (1) Subnet Mask
    Parameter Request List Item: (2) Time Offset
    Parameter Request List Item: (3) Router
    Parameter Request List Item: (4) Time Server
    Parameter Request List Item: (5) Name Server
    Parameter Request List Item: (6) Domain Name Server
    Parameter Request List Item: (12) Host Name
    Parameter Request List Item: (13) Boot File Size
    Parameter Request List Item: (15) Domain Name
    Parameter Request List Item: (17) Root Path
```

```
    Parameter Request List Item: (18) Extensions Path
    Parameter Request List Item: (22) Maximum Datagram Reassembly Size
    Parameter Request List Item: (23) Default IP Time-to-Live
    Parameter Request List Item: (28) Broadcast Address
    Parameter Request List Item: (40) Network Information Service Domain
    Parameter Request List Item: (41) Network Information Service Servers
    Parameter Request List Item: (42) Network Time Protocol Servers
    Parameter Request List Item: (43) Vendor-Specific Information
    Parameter Request List Item: (50) Requested IP Address
    Parameter Request List Item: (51) IP Address Lease Time
    Parameter Request List Item: (54) DHCP Server Identifier
    Parameter Request List Item: (58) Renewal Time Value
    Parameter Request List Item: (59) Rebinding Time Value
    Parameter Request List Item: (60) Vendor class identifier
    Parameter Request List Item: (66) TFTP Server Name
    Parameter Request List Item: (67) Bootfile name
    Parameter Request List Item: (97) UUID/GUID-based Client Identifier
    Parameter Request List Item: (128) DOCSIS full security server IP [TODO]
    Parameter Request List Item: (129) PXE - undefined (vendor specific)
    Parameter Request List Item: (130) PXE - undefined (vendor specific)
    Parameter Request List Item: (131) PXE - undefined (vendor specific)
    Parameter Request List Item: (132) PXE - undefined (vendor specific)
    Parameter Request List Item: (133) PXE - undefined (vendor specific)
    Parameter Request List Item: (134) PXE - undefined (vendor specific)
    Parameter Request List Item: (135) PXE - undefined (vendor specific)
Option: (97) UUID/GUID-based Client Identifier

Length: 17
    Client Identifier (UUID): 00000000-0000-0000-0000-44123456789a
Option: (94) Client Network Device Interface
    Length: 3
    Major Version: 3
    Minor Version: 16
Option: (93) Client System Architecture
    Length: 2
    Client System Architecture: EFI BC (7)
Option: (60) Vendor class identifier
    Length: 32
    Vendor class identifier: PXEClient:Arch:00007:UNDI:003016
Option: (255) End
    Option End: 255
```

Systems with UEFI

UEFI is a protocol known as Unified Extensible Firmware Interface (UEFI) Secure Boot. This was to be a modern replacement for the aging BIOS system and would help ensure boot-time malware couldn't be injected into a system.

The BIOS replacement, UEFI, requires a digital key installed for the OS to pass the UEFI firmware check to be able to boot. Mainstream Linux distributions like Red Hat, Ubuntu and Suse for example have purchased those keys so they have no problems with Secure Boot systems.

Without this digital key you generally still should be able to use Linux on a secure boot system. You can start with disabling the folowing in your BIOS:

```
                          Quickboot/Fastboot

                          Intel Smart Response Technology (ISRT)

                          FastStartUp (if you have Windows 8).
```

If you get a Secure boot or signature error, you need to disable Secure Boot. If your system is running Windows 7, you can enter the BIOS by entering the keyboard key required to enter the BIOS settings and disable Secure Boot. If the system comes with Windows 8 you will need to boot into Windows and choose to do an Advanced startup. This should allow you to enter the BIOS and disable Secure Boot. Not: Sometimes a BIOS is able to run in EFI or legacy mode. If your system allows this you should not have any problems installing Linux

Booting with Systemd-boot

Systemd comes with Systemd-boot. This is intended for use with EFI systems. It can only start EFI executables such as the Linux kernel EFISTUB, UEFI Shell, GRUB and the Windows Boot Manager. Systemd-boot is managed with the **bootctl** command. systemd-boot requires an EFI System Partition (ESP), preferably mounted on `/boot`. The ESP must contain the EFI binaries. Further information and examples can be found at https://www.freedesktop.org/software/systemd/man/index.html

Booting with Das U-boot

Das U-boot "the Universal Boot Loader" is an open source, primary boot loader aimed at embedded devices. It is used to package the instructions to boot the kernel code. It is supporting many computer architectures, including 68k, ARM, AVR32, Blackfin, MicroBlaze, MIPS, Nios, SuperH, PPC and x86. U-Boot can be split into stages if there are size restraints. U-Boot requires explicit commands as to where the memory addresses are to copy the kernel, ramdisk, etc data to opposed to other bootloaders which automatically choose the memory locations. Due to the U-Boot commands being low-level, booting a kernel requires multiple steps. This allows U-Boot to be very flexible. U-Boot can boot from on board storage, the network and even serial ports.

Questions and answers

System Startup

1. *Name all four user-mode processes, one of which the kernel will try to execute as the final step in the boot process.*

   In the final step in the boot process, the kernel will try to execute one of the following programs as the first user-mode process: **/sbin/init**, **/etc/init**, **/bin/init** and **/bin/sh**. First user-mode process [50]

2. *The parent of all processes init can be found running in one of eight runlevels. What type of levels are the runlevels 2, 3, 4, and 5?*

   These runlevels are multi-user runlevels. Multi-user runlevels [50]

3. *What does it mean when the action field in the file `/etc/inittab` contains the string **wait**?*

   The process specified in the fourth field of the same line will be started just once when the specified runlevel is entered and **init** will wait for its termination. The file /etc/inittab [51]

4. *What type of files do the directories `/etc/rc*`contain?*

   These directories contain symbolic links to scripts in `/etc/init.d` starting with either an **S** or a **K** which will call the script with either a start or a stop parameter. The /etc/init.d/rc script [52]

5. *What is the purpose of the Linux Standard Base (LSB) standard?*

   It ensures that a program compiled in an LSB compatible environment will run on any distribution that supports the LSB standard (within a certain processor architecture). The LSB standard [56]

6. *What is chain-loading?*

   Chain-loading implies that **GRUB** will be used to boot the system, and in turn will load and run the proprietary systems bootloader, which then boots the operating system. GRUB chain-loading [60]

7. *What is the purpose of the **grub-install** command?*

   The **grub-install** command is used to install stage 1 to either the MBR or within a partition. The command grub-install [63]

8. *Which command enables you to switch to a different runlevel?*

   This is done via the **telinit** command. The command telinit [64]

9. *During boot a filesystem check fails. You read something like "fsck failed. Please repair manually" followed by "CONTROL-D will exit from this shell and continue system startup". How do you proceed?*

   If you do not press CTRL-D but enter the root password, you will 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. When fsck is started but fails [64]

Chapter 3

Filesystem and Devices (203)

This topic has a weight of 9 points and contains the following three objectives:

Objective 203.1; Operating the Linux filesystem (4 points) Candidates should be able to properly configure and navigate the standard Linux filesystem. This objective includes configuring and mounting various filesystem types.

Objective 203.2; Maintaining a Linux filesystem (3 points) Candidates should be able to properly maintain a Linux filesystem using system utilities. This objective includes manipulating standard filesystems and monitoring SMART devices.

Objective 203.3; Creating and configuring filesystem options (2 points) Candidates should be able to configure automount filesystems using AutoFS. This objective includes configuring automount for network and device filesystems. Also included is creating filesystems for devices such as CD-ROMs and a basic feature knowledge of encrypted filesystems.

Operating The Linux Filesystem (203.1)

Candidates should be able to properly configure and navigate the standard Linux filesystem. This objective includes configuring and mounting various filesystem types.

Resources: LinuxRef07 , Wirzenius98 .

Key Knowledge Areas

- The concept of the `fstab` configuration
- Tools and utilities for handling SWAP partitions and files
- Use of UUIDs for identifying and mounting file systems
- Understanding of systemd mount units

Terms and Utilities

- `/etc/fstab`
- `/etc/mtab`
- `/proc/mounts`
- **mount** and **umount**
- **sync**

- **swapon**
- **swapoff**
- **blkid**

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 came into being. These were often small things such as the location (or placement) of certain configuration files, but this resulted in difficulties porting software from host to host.

To equalize these differences a file standard was developed. This, to date, is an evolving process resulting in a fairly static model for the Linux file hierarchy. This filesystem hierarchy is standardized in the filesystem hierarchy standard. The current version is 2.3. More information and documentation on the FHS can be found at *Filesystem Hierarchy Standard homepage* . See also the section on the FHS standard.

The top level of the Linux file hierarchy is referred to as the root (or /). The root directory typically contains several other directories. An overview was already presented in the section that discusses the contents of the root file system. A recap:

`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 home directories
`lib/`	Main OS shared libraries and kernel modules
`lost+found/`	Storage directory for "recovered" files
`media/`	Mount point(s) for removable media like CD-ROM's, flash disks and floppies
`mnt/`	Temporary mount point for filesystems as needed by a system administrator
`opt/`	Reserved for the installation of large add-on application software packages
`proc/`	A "virtual" filesystem used by Linux systems to store information about the kernel, processes and current resource usage
`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/`	Shareable, read-only data, containing e.g. user commands, C programs header files and non-vital system binaries
`var/`	Variable data, usually machine specific. Includes spool directories for mail and news, administrative and logging data

Table 3.1: Recap of the Linux file hierarchy

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

Filesystems

A filesystem consists of methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organised 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 files are stored, or one might say "I am 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 those 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 a filesystem 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 initializing the partition. 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)

Note
Creating a filesystem on a device with an existing filesystem will cause all data on the old filesystem to be erased.

Mounting and Unmounting

Linux presents all filesystems as one directory tree. Hence to add a new device with a filesystem on it its filesystem needs to be made part of that one directory tree. The way this is done is by attaching the new filesystem under an existing (preferably empty) directory, which is part of the existing directory tree - the "**mount**" point.

To attach a new file system to the directory hierarchy you must mount its associated device file. First you will need to create the mount point; a directory where the device will be attached. As directories are part of a filesystem too the mount point exists

on a previously mounted device. It should be empty. If is is not 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). This type of security by obscurity is sometimes used to hide information from the casual onlooker.

To mount a device, use the mount command:

```
mount [options] 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 [options] -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 (IDE) 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 which devices you currently have mounted, simply type the command **mount**. Some sample output:

```
/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 shows which 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 usage. 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 that user programs have access to this information.

The information in the proc filesystem can also be used to see which 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
```

The difference between `/etc/mtab` and `/proc/mounts` is that `/etc/mtab` is the user space administration kept by **mount**, and `/proc/mounts` is the information kept by the kernel. The latter reflects the information in user space. Due to these different implementations the info in `/proc/mounts` is always up-to-date, while the info in `/etc/mtab` may become inconsistent.

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 /dev/fd0
```

or

```
umount /mnt
```

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 (e.g. 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.

The system needs to mount devices during boot. In true UNIX fashion, there is a file which governs the behaviour of mounting devices at boot time. In Linux, this file is `/etc/fstab`. Lines from the file use 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.

There's also a file called `/etc/mtab`. It lists the currently mounted partitions in fstab form.

Systemd Mount Units

Linux distributions that have adopted the systemd initialization system have an additional way of mounting filesystems. Instead of using the `fstab` file for persistent mounting, a filesystem can be configured using a mount unit file. This mount unit file holds the configuration details for systemd to persistently mount filesystems.

A systemd mount unit file has a specific naming convention. The file name refers to the absolute directory it will be mounted on and the file extension is `.mount`. For the name of the file the first and last forward slash (/) of the mount path it represents are removed and the remaining slashes are converted to a dash (-). So if, for example, a filesystem is mounted to the mount point `/home/user/data/` the mount unit file must be named `home-user-data.mount`

In the mount file three required sections are defined: `[Unit]`, `[Mount]` and `[Install]`. An example of a mount unit file `/etc/systemd/system/home-user-data.mount`:

```
[Unit]
Description=Data for User

[Mount]
What=/dev/sda2
Where=/home/user/data
Type=ext4
Options=defaults

[Install]
WantedBy=multi-user.target
```

To test the configuration reload the systemctl daemon by using the command **systemctl daemon-reload** and then manually start the mount unit file with the command **systemctl start** followed by the mount unit file. In our example that would be **systemctl start home-user-data.mount**. Next you can check if the filesystem was mounted correctly by getting the overview from **mount**. If everything works as expected make the filesystem mount persistent by enabling the mount unit file with the command **systemctl enable home-user-data.mount**.

Swap

Swap space in Linux is a partition or file that is used to move the contents of inactive pages of RAM to when RAM becomes full. 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 in doubt, you could use a swap file first, and 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. 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 at the beginning of the platters (outside rim, first cylinders). 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 may 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 may 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 report the total amount of swap space used:

```
$ free
total used free shared buffers cached
Mem: 127148 122588 4560 50 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 used by tmpfs (shmem in /proc/meminfo); 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
```

To disable a device or swap file, use the **swapoff** command:

```
# swapoff /dev/sda3
```

UUIDs

The term UUID stands for Universal Unique IDentifier. It's a 128 bit number that can be used to identify basically anything. Generating such UUIDs can be done using appropriate software. There are 5 various versions of UUIDs, all of them use a (pseudo)random element, current system time and some mostly unique hardware ID, for example a MAC address. Theoretically there is a very, very remote chance of an UUID not being unique, but this is seen as impossible in practice.

On Linux, support for UUIDs was started within the e2fsprogs package. With filesystems, UUIDs are used to represent a specific filesystem. You can for example use the UUID in `/etc/fstab` to represent the partition which you want to mount.

Usually, a UUID is represented as 32 hexadecimal digits, grouped in sequences of 8,4,4,4 and 12 digits, separated by hyphens. Here's what an fstab entry with a UUID specifier looks like:

```
UUID=652b786e-b87f-49d2-af23-8087ced0c828 / ext4 errors=remount-ro,noatime 0 1
```

You might be wondering about the use of UUID's in fstab, since device names work fine. UUIDs come in handy when disks are moved to different connectors or computers, multiple operating systems are installed on the computer, or other cases where device names could change while keeping the filesystem intact. As long as the filesystem does not change, the UUID stays the same.

Note the 'as long as the filesystem does not change'. This means, when you reformat a partition, the UUID will change. For example, when you use mke2fs to reformat partition `/dev/sda3`, the UUID will be changed. So, if you use UUIDs in `/etc/fstab`, you have to adjust those as well.

If you want to know the UUID of a specific partition, use **blkid /path/to/partition**:

```
# blkid /dev/sda5
/dev/sda5: UUID="24df5f2a-a23f-4130-ae45-90e1016031bc" TYPE="swap"
```

Note
It is possible to create a new filesystem and still make it have the same UUID as it had before, at least for 'ext' type filesystems.

```
# tune2fs /dev/sda5 -U 24df5f2a-a23f-4130-ae45-90e1016031bc
```

Note
On most Linux distributions you can generate your own UUIDs using the command **uuidgen**.

sync

To improve performance of Linux filesystems, many operations are done in filesystem buffers, stored in RAM. To actually flush the data contained in these buffers to disk, the **sync** command is used.

sync is called automatically at the right moment when rebooting or halting the system. You will rarely need to use the command yourself. **sync** might be used to force syncing data to an USB device before removing it from your system, for example.

sync does not have any operation influencing options, so when you need to, just execute "**sync**" on the command line.

Maintaining a Linux Filesystem (203.2)

Candidates should be able to properly maintain a Linux filesystem using system utilities. This objective includes manipulating standard filesystems and monitoring SMART devices.

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

Key Knowledge Areas

- Tools and utilities to manipulate ext2, ext3 and ext4.
- Tools and utilities to perform Btrfs operations, including subvolumes and snapshots.
- Tools and utilities to manipulate xfs.
- Awareness of ZFS.

Terms and Utilities

- **fsck (fsck.*)**
- **mkfs (mkfs.*)**
- **mkswap**
- **tune2fs**
- **dumpe2fs**
- **debugfs**
- **btrfs**
- **btrfs-convert**
- **xfs_info**, **xfs_check**, **xfs_repair**, **xfs_dump** and **xfs_restore**
- **smartd** and **smartctl**

Disk Checks

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.*)

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", or the maximum mount count is reached or the time between checks 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're 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(=options) 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.

mkfs (mkfs.*)

The **mk2fs** command is used to create a Linux filesystem. It can create different types of filesystems by specifying the `-t` *filesystem* or by giving the **mkfs.filesystem** command. Actually **mkfs** is a front-end for the several **mkfs.fstype** commands. Please read the mkfs man pages and section for more information. Creating Filesystems

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 suffix 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 a Linux utility to check for damaged sectors on a disk drive. It marks these sectors so that they are not used in the future and thus do not cause corruption of data. It is part of the e2fsprogs project.

It is strongly recommended that badblocks not be run directly but to have it invoked through the `-c` option in **e2fsck** or **mke2fs**. A commonly used option is:

-o output-file write the list of bad blocks to `output-file`.

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 options. To open in read-write mode, add the `-w` option. You may also want to include in the command line the device you wish 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 of **debugfs**, 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**.

ext4

Ext4 is the evolution of the most used Linux filesystem, Ext3. In many ways, Ext4 is a deeper improvement over Ext3 than Ext3 was over Ext2. Ext3 was mostly about adding journaling to Ext2, but Ext4 modifies important data structures of the filesystem such as the ones destined to store the file data. The result is a filesystem with an improved design, better performance, reliability, and features. Therefore converting ext3 to ext4 is not as straightforward and easy as it was converting ext2 to ext3.

To creating ext4 partitions from scratch, use:

```
mkfs.ext4 /dev/sdxY
```

Note
Tip: See the mkfs.ext4 man page for more options; edit `/etc/mke2fs.conf` to view/configure default options.

Be aware that by default, mkfs.ext4 uses a rather low bytes-per-inode ratio to calculate the fixed amount of inodes to be created.

Note
Note: Especially for contemporary HDDs (750 GB+) this usually results in a much too large inode number and thus many likely wasted GB. The ratio can be set directly via the -i option; one of 6291456 resulted in 476928 inodes for a 2 TB partition.

For the rest ext4 can be manipulated using all the same tools that are available for ext2/ext3 type of filesystems like badblocks, dumpe2fs, e2fsck and tune2fs.

btrfs

Btrfs (abbreviation for: *BTree File System*)is a new copy on write (CoW) filesystem for Linux aimed at implementing advanced features while focusing on fault tolerance, repair and easy administration. Jointly developed at Oracle, Red Hat, Fujitsu, Intel, SUSE, STRATO and many others, Btrfs is licensed under the GPL and open for contribution from anyone. Btrfs has several features characteristic of a storage device. It is designed to make the file system tolerant of errors, and to facilitate the detection and repair of errors when they occur. It uses checksums to ensure the validity of data and metadata, and maintains snapshots of the file system that can be used for backup or repair. The core datastructure used by btrfs is the *B-Tree* - hence the name.

Note
Btrfs is still under heavy development, but every effort is being made to keep the filesystem stable and fast. Because of the speed of development, you should run the latest kernel you can (either the latest release kernel from kernel.org, or the latest -rc kernel.

As of the beginning of the year 2013 Btrfs was included in the default kernel and its tools (btrfs-progs) are part of the default installation. GRUB 2, mkinitcpio, and Syslinux have support for Btrfs and require no additional configuration.

The main Btrfs features available at the moment include:

- Extent based file storage
- 2^64 byte == 16 EiB maximum file size
- Space-efficient packing of small files
- Space-efficient indexed directories
- Dynamic inode allocation
- Writable snapshots, read-only snapshots
- Subvolumes (separate internal filesystem roots)
- Checksums on data and metadata (crc32c)
- Compression (zlib and LZO)
- Integrated multiple device support
- File Striping, File Mirroring, File Striping+Mirroring, Striping with Single and Dual Parity implementations
- SSD (Flash storage) awareness (TRIM/Discard for reporting free blocks for reuse) and optimizations (e.g. avoiding unnecessary seek optimizations, sending writes in clusters, even if they are from unrelated files. This results in larger write operations and faster write throughput)
- Efficient Incremental Backup
- Background scrub process for finding and fixing errors on files with redundant copies
- Online filesystem defragmentation
- Offline filesystem check
- Conversion of existing ext3/4 file systems
- Seed devices. Create a (readonly) filesystem that acts as a template to seed other Btrfs filesystems. The original filesystem and devices are included as a readonly starting point for the new filesystem. Using copy on write, all modifications are stored on different devices; the original is unchanged.
- Subvolume-aware quota support
- Send/receive of subvolume changes

- Efficient incremental filesystem mirroring

The most notable (unique) btrfs features are:

RAID functionality A Btrfs filesystem provides support for integrated RAID functionality, and can be created on top of many devices. More devices can be added after the filesystem is created. By default, metadata will be mirrored across two devices and data will be striped across all of the devices present. If only one device is present, metadata will be duplicated on that one device.

Btrfs can add and remove devices online, and freely convert between RAID levels after the filesystem is created. Btrfs supports raid0, raid1, raid10, raid5 and raid6, and it can also duplicate metadata on a single spindle. When blocks are read in, checksums are verified. If there are any errors, Btrfs tries to read from an alternate copy and will repair the broken copy if the alternative copy succeeds.

mkfs.btrfs will accept more than one device on the command line. It has options to control the raid configuration for data (-d) and metadata (-m). Valid choices are raid0, raid1, raid10 and single. The option -m single means that no duplication of metadata is done, which may be desired when using hardware raid. here some examples on creating the filesystem.

```
# Create a filesystem across four drives (metadata mirrored, linear data allocation)
mkfs.btrfs /dev/sdb /dev/sdc /dev/sdd /dev/sde

# Stripe the data without mirroring
mkfs.btrfs -d raid0 /dev/sdb /dev/sdc

# Use raid10 for both data and metadata
mkfs.btrfs -m raid10 -d raid10 /dev/sdb /dev/sdc /dev/sdd /dev/sde

# Don't duplicate metadata on a single drive (default on single SSDs)
mkfs.btrfs -m single /dev/sdb
```

After filesystem creation it can be mounted like any other filesystem. To see the devices used by the filesystem you can use the follwing command:

```
butterfs filesystem show
label:  none  uuid: c67b1e23-887b-4cb9-b037-5958f6c0a333
  total devices 2 FS bytes used 383.00KiB
  devid 1 size 4.00 GiB used 847.12MiB path /dev/sdb
  devid 2 size 4.00 GiB used 837.12MiB path /dev/sdc
Btrfs v4.8.5
```

Snapshotting Btrfs's snapshotting is simple to use and understand. The snapshots will show up as normal directories under the snapshotted directory, and you can cd into it and walk around there as you would in any directory.

By default, all snapshots are writeable in Btrfs, but you can create read-only snapshots if you choose so. Read-only snapshots are great if you are just going to take a snapshot for a backup and then delete it once the backup completes. Writeable snapshots are handy because you can do things such as snapshot your file system before performing a system update; if the update breaks your system, you can reboot into the snapshot and use it like your normal file system. When you create a new Btrfs file system, the root directory is a subvolume. Snapshots can only be taken of subvolumes, because a subvolume is the representation of the root of a completely different filesystem tree, and you can only snapshot a filesystem tree.

The simplest way to think of this would be to create a subvolume for `/home`, so you could snapshot `/` and `/home` independently of each other. So you could run the following command to create a subvolume:

```
btrfs subvolume create /home
```

And then at some point down the road when you need to snapshot `/home` for a backup, you simply run:

```
btrfs subvolume snapshot /home/ /home-snap
```

Once you are done with your backup, you can delete the snapshot with the command

```
btrfs subvolume delete /home-snap/
```

The hard work of unlinking the snapshot tree is done in the background, so you may notice I/O happening on a seemingly idle box; this is just Btrfs cleaning up the old snapshot. If you have a lot of snapshots or don't remember which directories you created as subvolumes, you can run the command:

```
# btrfs subvolume list /mnt/btrfs-test/
ID 267 top level 5 path home
ID 268 top level 5 path snap-home
ID 270 top level 5 path home/josef
```

This doesn't differentiate between a snapshot and a normal subvolume, so you should probably name your snapshots consistently so that later on you can tell which is which.

Subvolumes A subvolume in btrfs is not the same as an LVM logical volume, or a ZFS subvolume. With LVM, a logical volume is a block device in its own right; this is not the case with btrfs. A btrfs subvolume is not a block device, and cannot be treated as one.

Instead, a btrfs subvolume can be thought of as a POSIX file namespace. This namespace can be accessed via the top-level subvolume of the filesystem, or it can be mounted in its own right. So, given a filesystem structure like this:

```
toplevel
  `--- dir_a                * just a normal directory
  |         `--- p
  |         `--- q
  `--- subvol_z             * a subvolume
      `--- r
      `--- s
```

the root of the filesystem can be mounted, and the full filesystem structure will be seen at the mount point; alternatively the subvolume can be mounted (with the **mount** option `subvol=subvol_z`), and only the files `r` and `s` will be visible at the mount point.

A btrfs filesystem has a default subvolume, which is initially set to be the top-level subvolume. It is the default subvolume which is mounted if no subvol or subvolid option is passed to mount. Changing the default subvolume with *btrfs subvolume default* will make the top level of the filesystem inaccessible, except by use of the `subvolid=0` mount option.

Space-efficient indexed directories Directories and files look the same on disk in Btrfs, which is consistent with the UNIX way of doing things. The ext file system variants have to pre-allocate their inode space when making the file system, so you are limited to the number of files you can create once you create the file system.

With Btrfs we add a couple of items to the B-tree when you create a new file, which limits you only by the amount of metadata space you have in your file system. If you have ever created thousands of files in a directory on an ext file system and then deleted the files, you may have noticed that doing an ls on the directory would take much longer than you'd expect given that there may only be a few files in the directory.

You may have even had to run this command:

```
e2fsck -D /dev/sda1
```

to re-optimize your directories in ext. This is due to a flaw in how the directory indexes are stored in ext: they cannot be shrunk. So once you add thousands of files and the internal directory index tree grows to a large size, it will not shrink back down as you remove files. This is not the case with Btrfs.

In Btrfs we store a file index next to the directory inode within the file system B-tree. The B-tree will grow and shrink as necessary, so if you create a billion files in a directory and then remove all of them, an ls will take only as long as if you had just created the directory.

Btrfs also has an index for each file that is based on the name of the file. This is handy because instead of having to search through the containing directory's file index for a match, we simply hash the name of the file and search the B-tree for this hash value. This item is stored next to the inode item of the file, so looking up the name will usually read in the same block that contains all of the important information you need. Again, this limits the amount of I/O that needs to be done to accomplish basic tasks.

mkswap

mkswap sets up a Linux swap area on a device or in a file. (After creating the swap area, you need to invoke the **swapon** command to start using it. Usually swap areas are listed in `/etc/fstab` so that they can be taken into use at boot time by a **swapon** `-a` command in some boot script.) See Swap file usage [80]

xfs_info

xfs_info shows the filesystem geometry for an XFS filesystem. xfs_info is equivalent to invoking xfs_growfs with the `-n` option.

xfs_check

xfs_check checks whether an XFS filesystem is consistent. It is needed only when there is reason to believe that the filesystem has a consistency problem. Since XFS is a Journalling filesystem, which allows it to retain filesystem consistency, there should be little need to ever run **xfs_check**.

xfs_repair

xfs_repair repairs corrupt or damaged XFS filesystems. xfs_repair will attempt to find the raw device associated with the specified block device and will use the raw device instead. Regardless, the filesystem to be repaired must be unmounted, otherwise, the resulting filesystem may become inconsistent or corrupt.

smartmontools: smartd and smartctl

Two utility programs, *smartctl* and *smartd* (available when the *smartmontools* package is installed) can be used to monitor and control storage systems using the *Self-Monitoring, Analysis and Reporting Technology System (SMART)*. SMARTis built into most modern ATA and SCSI harddisks and solid-state drives. The purpose of SMARTis to monitor the reliability of the hard drive and predict drive failures, and to carry out different types of drive self-tests.

*smartd*is a daemon that will attempt to enable SMARTmonitoring on ATA devices and polls these and SCSI devices every 30 minutes (configurable), logging SMARTerrors and changes of SMARTAttributes via the SYSLOG interface. smartd can also be configured to send email warnings if problems are detected. Depending upon the type of problem, you may want to run self-tests on the disk, back up the disk, replace the disk, or use a manufacturer's utility to force reallocation of bad or unreadable disk sectors.

smartd can be configured at start-up using the configuration file /usr/local/etc/smartd.conf. When the USR1 signal is sent to smartd it will immediately check the status of the disks, and then return to polling the disks every 30 minutes. Please consult the manual page for smartd for specific configuration options.

The *smartctl* utility controls the SMART system. It can be used to scan devices and print info about them, to enable or disable SMART on specific disks, to configure what to do when (imminent) errors are detected. Please consult the smartctl manual page for details.

ZFS: zpool and zfs

ZFS, currently owned by Oracle Corporation, was developed at Sun Microsystems as a next generation filesystem aimed at near infinite scalability and free from traditional design paradigms. Only Ubuntu currently offers kernel integration for ZFS on Linux. Other Linux distributions can make use of ZFS through userspace with the aid of Fuse.

The main ZFS features available at the moment include:

- High fault tolerance and data integrity
- Near unlimited storage due to 128 bit design

- Hybrid volume/filesystem management with RAID capabilities
- Compression/deduplication of data
- Data snapshots
- Volume provisioning (zvols)
- Ability to use different devices for caching and logging
- Ability to use delegate administrative rights to unprivileged users

The high fault tolerance and data integrity design makes it unneccesary to have a command like fsck available. ZFS works in transactions in where the ueberblock is only updated if everything was completed. Copies of previous uberblocks (128) are being kept in a round robin fashion. The so called vdev labels, which identify the disks used in a zfs pool, also have multiple copies: 2 at the beginning of the disk and 2 at the end. Periodic scrubbing of pools can reveal data integrity issues, and if a pool is equipped with more than one device, can repair it on the fly.

ZFS allows for additional checksumming algoritms of blocks and can have multiple copies of those blocks stored. This can be configured per ZFS filesystem.

ZFS pools can be considered the logical volume managent and can consist out of single or multiple disks. The pools can have seperate cache and logging devices attached so that reading/writing is offloaded to faster devices. Having multiple disks in a pool allows for on the fly data recovery.

A typical simple ZFS pool:

```
$ sudo zpool list -v
NAME   SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  1.98G   448K  1.98G         -     0%     0%  1.00x  ONLINE  -
 sdb  1.98G   448K  1.98G         -     0%     0%
```

Status of the pool:

```
$ sudo zpool status -v
  pool: tank
 state: ONLINE
  scan: none requested
config:

  NAME        STATE     READ WRITE CKSUM
  tank        ONLINE       0     0     0
    sdb       ONLINE       0     0     0
  errors: No known data errors
```

History overview for ZFS pools:

```
$ sudo zpool history
History for 'tank':
2017-01-11.11:32:50 zpool create -f -m /tank tank /dev/sdb
2017-01-11.12:37:44 zpool scrub tank
```

The ZFS pool is used as backing for one or more ZFS filesystems. By default the pool itself has a single filesystem named after the pool. ZFS filesystems are flexible in size and allow for customisation of various attributes like compression, size reservation, quota, block copies and so on. A full set of attributes can be seen with **zfs get all** and optionally a zfs filesystem, snapshot or volume.

A typical ZFS filesystem listing

```
$ sudo zfs list
NAME   USED  AVAIL  REFER  MOUNTPOINT
tank   242K  1.92G    19K  /tank
```

Create a ZFS filesystem "documents" and use compression

```
$ sudo zfs create -o compression=on tank/documents
$ sudo zfs list tank/documents
NAME              USED  AVAIL  REFER  MOUNTPOINT
tank/documents     19K  1.92G    19K  /tank/documents
```

or

```
$ sudo zfs create tank/documents
$ sudo zfs set compression=on tank/documents
$ sudo zfs list tank/documents
NAME              USED  AVAIL  REFER  MOUNTPOINT
tank/documents     19K  1.92G    19K  /tank/documents
```

With some data in `/tank/documents` compression ratio can be seen with

```
$ sudo zfs get compressratio tank/documents
NAME             PROPERTY       VALUE  SOURCE
tank/documents   compressratio  1.68x  -
```

Creating a backup of `/tank/documents` can be done instantaniously and takes no space until `/tank/documents`'s content changes. The contents of the snapshot can be accessed through the `.zfs/snapshot` directory of that ZFS filesystem.

```
$ sudo zfs snap tank/documents@backup
$ sudo zfs list -t snapshot
NAME                     USED  AVAIL  REFER  MOUNTPOINT
tank/documents@backup      0      -  45.0M  -
```

Creating And Configuring Filesystem Options (203.3)

Candidates should be able to configure automount filesystems using AutoFS. This objective includes configuring automount for network and device filesystems. Also included is creating filesystems for devices such as CD-ROMs.

Resources: Don99; Nielsen98; Truemper00.

Key Knowledge Areas

- autofs configuration files
- UDF and ISO9660 tools and utilities
- awareness of CD-ROM filesystems (UDF, ISO9660, HFS)
- awareness of CD-ROM filesystem extensions (Joliet, Rock Ridge, El Torito)
- basic feature knowledge of encrypted filesystems

Terms and Utilities

- `/etc/auto.master`
- `/etc/auto.[dir]`
- **mkisofs**
- **dd**
- **mke2fs**

Autofs and automounter

Automounting is the process in which 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 contains lines with three whitespace separated fields. The first field lists the directory in which the mounts will be done. The second field lists the filename in which we have placed configuration data for the devices to be mounted, the "supplemental" files. The last field specifies options. In our example we specified a timeout. After the timeout period lapses, the automount daemon will unmount the devices specified in the supplemental file.

The configuration data in the supplemental files may consist of multiple lines and span multiple devices. The filenames and path to the supplemental files may be choosen freely. Each line in a supplemental file contains three fields:

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

The first value is the "pseudo" directory. If the device is mounted, a directory of that name will appear and you can change into it to explore the mounted filesystem. 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 configuration files are reread when the automounter is reloaded

```
/etc/init.d/autofs reload
```

Please note that autofs does NOT reload nor restart if the mounted directory (eg: `/home`) is busy. Every entry in `auto.master` starts it's own daemon.

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` does not exist yet. Only when you directly try to access that directory the automounter will mount it.

Each device should have its own supplementary file. So, for example, configuration data for the floppy drive and that of the cdrom drive should not be combined into the same supplementary file. Each definition in the `/etc/auto.master` file results in spawning its own **automount** daemon. If you have several devices running under control of the same automount daemon - which is legit - but one of the devices fails, the daemon may hang or be suspended and other devices it controls might not be mounted properly either. Hence it is good practice to have every device under control ofits own automount daemon and so there should be just one device per supplementary file per entry in the `/etc/auto.master` file.

Automount with systemd To initiate automount with systemd a unit file should be created in `/etc/systemd/system`. The unit file should be named after the mount point. In this example the file is named: mnt.mount because the mount point is /mnt.

```
# cat /etc/systemd/systemd/mnt.mount
[Unit]
Description=My new file system

[Mount]
What=/dev/sdb1
Where=/mnt
Type=ext4
Options=
```

```
[Install]
WantedBy=multi-user.target
```

After creating the unit file it should be activated with the command:

```
# systemctl start mnt.mount
```

Verify the activated mount point:

```
# mount | grep sdb1
/dev/sdb1 on /mnt type ext4 (rw,relatime,data=ordered)
```

To activate the auto mounting on startup use the command:

```
# systemctl enable mnt.mount
```

Initiate a reboot and verify if the partition is mounted. More information about mounting with systemd can be found in the man page of systemd.mount(5).

Autofs with systemd To enable autofs with systemd a unit file with the extension `.automount` should be created in `/etc/systemd/system`. The information in that file is being controlled and supervised by systemd. The Automount units should be named after the directions they control. In this example the name of the automount unit configuration file is: `mnt.automount`.

```
# cat /etc/systemd/system/mnt.automount
[Unit]
Description=My new automounted file system.

[Automount]
Where=/mnt

[Install]
WantedBy=multi-user.target
```

Activate the autofs automount unit.

```
# systemctl status mnt.automount
● mnt.automount - My new automounted file system.
Loaded: loaded (/etc/systemd/system/mnt.automount; disabled; vendor preset: disabled)
  Active: inactive (dead)
    Where: /mnt

# systemctl enable mnt.automount
Created symlink from /etc/systemd/system/multi-user.target.wants/mnt.automount to /etc/ ↩
    systemd/system/mnt.automount.

# systemctl start mnt.automount

# systemctl status mnt.automount
● mnt.automount - My new automounted file system.
Loaded: loaded (/etc/systemd/system/mnt.automount; enabled; vendor preset: disabled)
  Active: active (waiting) since Thu 2016-12-29 14:01:57 AST; 1min 2s ago
    Where: /mnt

Dec 29 14:01:57 snow systemd[1]: Set up automount My new automounted file system..
```

Once it has been started the mount output shows the mount point enabled with autofs.

```
# mount |grep mnt
systemd-1 on /mnt type autofs (rw,relatime,fd=25,pgrp=1,timeout=0,minproto=5,maxproto=5, ↩
    direct)
```

The command **lsblk** shows that that disk is not mounted.

```
# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   30G  0 disk
&#x251c;&#x2500;sda1   8:1    0 28.8G  0 part /
&#x251c;&#x2500;sda2   8:2    0    1K  0 part
&#x2514;&#x2500;sda5   8:5    0  1.3G  0 part [SWAP]
sdb      8:16   0    1G  0 disk
&#x2514;&#x2500;sdb1   8:17   0 1023M  0 part
```

Let's do a file listing of the /mnt directory.

```
# ls /mnt
lost+found
```

Then execute the command **lsblk** again to verify if partition sdb1 is mounted on /mnt.

```
# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   30G  0 disk
&#x251c;&#x2500;sda1   8:1    0 28.8G  0 part /
&#x251c;&#x2500;sda2   8:2    0    1K  0 part
&#x2514;&#x2500;sda5   8:5    0  1.3G  0 part [SWAP]
sdb      8:16   0    1G  0 disk
&#x2514;&#x2500;sdb1   8:17   0 1023M  0 part /mnt
```

The **mount** command shows the same.

```
# mount | grep /mnt
systemd-1 on /mnt type autofs (rw,relatime,fd=25,pgrp=1,timeout=0,minproto=5,maxproto=5, ↩
    direct)
/dev/sdb1 on /mnt type ext4 (rw,relatime,data=ordered)
```

Autofs combined with systemd is now working. See the manpage of systemd.automount(5) for more configuration options.

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/
```

The option `-r` sets the permissions of all files on the CD to be public readable and enables Rock Ridge 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 Rock Ridge 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. Another extention is El Torito, which is used to create bootable CD-ROM filesystems.

Besides the ISO9660 filesystem as created by **mkisofs** there is the UDF (Universal Disk Format) filesystem. The Optical Storage Technology Association standardized the UDF filesystem to form a common filesystem for all (read-only and re-writable) optical media. It was intended to replace the ISO9660 filesystem.

Tools to create and maintain a UDF filesystem are:

mkudffs Creates a new UDF filesystem. Can be used on hard disks as well as on CD-R(W).

udffsck Used to check the integrity and correct errors on UDF filesystems.

wrudf This command is used for maintaining UDF filesystems. It provides an interactive shell with operations on existing UDF filesystems: cp, rm, mkdir, rmdir, ls,.... etc.

cdrwtool The **cdrwtool** provides facilities to manage CD-RW drives. This includes formating new disks, setting the read and write speeds, etc.

These tools are part of the UDFtools package.

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 650MB-sized 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.

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 the CD may be corrupted. It is easy to unintentionally interrupt the data stream, for example by deleting a very large file. Say you delete an old 650 Mb CD-image - the kernel must update 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 will 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.

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

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

# same as above, but shorter:
# 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 whose names actually make sense: `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:

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

# echo $IMG_SIZE

# [ "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**).

Encrypted file systems

Linux has native filesystem encryption support. You can choose from a number of symmetric encryption algorithms to encrypt your filesystem with, namely: Twofish, Advanced Encryption Standard (AES) also known as Rijndael, Data Encryption Standard (DES) and others. Twofish was also an AES candidate like Rijndael. Due performance reasons Rijndael was selected as the new

AES standard. Twofish supports 128 bit block size and keys up to 256 bits. DES is the predecessor of the AES standard and is now considered as insecure because of the small key size. With current computing power it is possible to brute force 56 bits (+8 parity bits) DES keys in a relatively short time frame. AES uses a block size of 128 bits and a key size of 128, 192 or 256 bits. For many years 128 bits key size was sufficient but with the introduction of quantum computers the U.S. National Security Agency issued guidance for data classification up to Top Secret with 256 bits keys. Intel introduced in 2010 the Advanced Encryption Standard New Instructions (AES-NI) set. This new instruction set performs the encryption and decryption completely in hardware which helps to lower the risk of side-channel attacks and greatly improve AES performance. To check if your CPU supports the AES-NI instruction set use the command: **grep aes /proc/cpuinfo** . You can check AES-NI kernel support with the command: **sort -u /proc/crypto | grep module** and load the driver with the command: **modprobe aesni-intel** as root.

Device Mapper As of linux 2.6 it is possible to use the devicemapper, a generic linux framework to map one block device to another. Devicemapper is used for software RAID and LVM. It is used as the filter between the filesystem on a virtual blockdevice and the encrypted data to be written to a hard disk. This enables the filesystem to present itself decrypted while everything read and written will be encrypted on disk. A virtual block device is created in `/dev/mapper`, which can be used as any other block device. All data to and from it goes to an encryption or decryption filter before being mapped to another blockdevice.

Device Mapper crypt (dm-crypt) Device mapper crypt provides a generic way for transparent encryption of block devices by using the kernel API and can be used in combination with RAID or LVM volumes. When the user creates a new block device he can specify several options: symmetric cipher, encryption mode, key and iv generation mode. Dm-crypt does not store any information in a header like LUKS does. After encrypting the disk it will be indistinguishable from a disk with random data. This means that the existence of of encrypted files deniable in the sense that it can not be proven that encrypted data exists. The user should keep track of the options that are used in the dm-crypt setup otherwise it could lead to data loss since no metadata is available. Only one encryption key can be used to encrypt and decrypt block devices and no master key can be used. Once the password of a encrypted block device is lost there is no possibility to recover the data. Dm-crypt should only be used by advanced users. Regular users should use LUKS for disk encryption.

Linux Unified Key Setup (LUKS) LUKS is a standard utility on all generic linux distributions. It provides disk encryption for different type of volumes (plain dm-crypt volumes, LUKS volumes, etc.). It offers compatibility amongst distributions and provides secure management of user passwords. It also provides storage of cryptography options including the master key in the partition header enabling seamlessly transport or data migration. LUKS also provides up to 8 different keys per LUKS partition to enable key escrow (usage of keys per meaning). By creating encrypted partitions on different Linux distributions the default settings may vary but the settings are sufficient enough to protect the data on the volume(s). LUKS is the preferred method for data protection by regular users.

Example with dm-crypt This is an example to set up an encrypted filesystem. All relevant modules should be loaded at boot time:

```
# echo aes >> /etc/modules
# echo dm_mod >> /etc/modules
# echo dm_crypt >> /etc/modules
# modprobe -a aes dm_mod dm_crypt
```

Create the device mapperblock device and use (for example) hda3 for it. Choose your password using: **cryptsetup -y create crypt /dev/hda3** Map the device:

```
# echo "crypt /dev/hda3 none none" >> /etc/crypttab
# echo "/dev/mapper/crypt /crypt reiserfs defaults 0 1" >> /etc/fstab
```

Make a filesystem:

```
# mkfs.reiserfs /dev/mapper/crypt
```

Now mount your encrypted filesystem. You will be prompted for the password you chose with cryptsetup. You will be asked to provide it at every boot:

```
# mkdir /crypt
# mount /crypt
```

Example with LUKS This is an example to create an 512 MiB encrypted LUKS container in a linux environment.

```
# dd if=/dev/urandom of=/root/encrypted bs=1M count=512
```

Format the sparse file with LUKS and provide a passphrase:

```
# cryptsetup -y luksFormat encrypted

WARNING!
========
This will overwrite data on encrypted irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase:
Verify passphrase:
```

Open the container, provide a name and enter the passphrase:

```
# cryptsetup luksOpen encrypted encrypted
Enter passphrase for encrypted:
```

Create a filesystem on the unencrypted container:

```
mkfs.ext4 /dev/mapper/encrypted
mke2fs 1.43.3 (04-Sep-2016)
Creating filesystem with 522240 1k blocks and 130560 inodes
Filesystem UUID: 9fb28cc0-5a3e-4e0b-b3cc-3cbd6cf3c8f4
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729, 204801, 221185, 401409

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
```

Mount the encrypted container for usage:

```
# mount /dev/mapper/encrypted /mnt
```

Encrypted container as filesystem:

```
# df -h
Filesystem             Size  Used Avail Use% Mounted on
udev                    10M     0   10M   0% /dev
tmpfs                  792M   26M  767M   4% /run
/dev/sda1               29G   16G   11G  60% /
tmpfs                  2.0G  400K  2.0G   1% /dev/shm
tmpfs                  5.0M     0  5.0M   0% /run/lock
tmpfs                  2.0G     0  2.0G   0% /sys/fs/cgroup
tmpfs                  396M   16K  396M   1% /run/user/133
tmpfs                  396M   20K  396M   1% /run/user/0
/dev/mapper/encrypted  486M  2.3M  455M   1% /mnt
```

Content of /mnt as any ext4 filesystem:

```
# ls /mnt
lost+found
```

Unmount the encrypted container after usage:

```
# umount /mnt
```

Close the encrypted container:

```
# cryptsetup luksClose /dev/mapper/encrypted
```

Questions and answers

Filesystem and Devices

1. *Why was it necessary to develop the Linux File Hierarchy?*

   The location of certain files and utilities not being standardized led to problems with development and upgrading between various distributions of Linux. Linux File Hierarchy [77]

2. *Does the inode part of the UNIX filesystem structure contain the name of a file?*

   No, an inode contains all the information on a file, except its name. UNIX filesystem structure [78]

3. *What is the fastest way to erase all data in an ext2 filesystem mounted at `/mnt`?*

   The command **mkfs -t ext2 /mnt** will erase all data in the `/mnt` filesystem by creating a new, empty filesystem. Part of the data may still be recoverable, of course. Erasing all data in a filesystem [78]

4. *Which file in the pseudo directory `/proc` contains basically the same information about swap as does the command **free**?*

   The file `/proc/meminfo` contains similar information. Swap information [81]

5. *What would you type before rebooting a system to ensure that the **ext** filesystems are not checked during boot?*

   Use e.g. **tune2fs -c 5 -C 0 /dev/hda1** as well as for all other devices, in order to postpone filesystem checking for 5 remounts. Use tune2fs [85]

6. *What switches would you use in order to run **debugfs** in read-only mode?*

   Debugfs operates in read-only mode by default, therefore no switches are needed. Use debug2fs [86]

7. *What is the main purpose of auto-mounting?*

   It avoids the necessity of having to use the **mount** command in a number of situations. Mounting and unmounting is done automatically upon accessing the directory upon which the mount is to be done. This is useful in networked environments and for removable devices, such as USB attached disks and CD-ROMs. Autofs and automounter [93]

8. *What happens if you combine the floppy drive and the cdrom drive into the same supplementary (**autofs**) file?*

   Each file will have only one **automount** program running for it, so if one entry fails, the other will not work either. One supplementary file per entry [93]

9. *Which two commands are used for creating a mountable CD-ROM?*

   mkisofs is used to create the ISO9660 image and **cdrecord** is used to write that image to a CD-ROM. CD-ROM filesystem [95]

Chapter 4

Advanced Storage Device Administration (204)

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

Objective 204.1; Configuring RAID (3 points) Candidates should be able to configure and implement software RAID. This objective includes using and configuring RAID 0, 1 and 5.

Objective 204.2; Adjusting Storage Device Access (2 points) Candidates should be able to configure kernel options to support various drives. This objective includes software tools to view and modify hard disk settings.

Objective 204.3; Logical Volume Manager (3 points) Candidates should be able to create and remove logical volumes and physical volumes. This objective includes snapshots, and resizing logical volumes.

Configuring RAID (204.1)

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

Resources: LinuxRef01; Robbins01; Bar00; manpages for **mdadm** and `mdadm.conf`.

Key Knowledge Areas

- Software raid configuration files and utilities

Terms and Utilities

- **mdadm**
- `mdadm.conf`
- **fdisk**
- `/proc/mdstat`

What is RAID?

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

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

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.

Some of the concepts used in this chapter are important factors in deciding which level of RAID to use in a particular situation. Parity is a concept used to add redundancy to storage. A parity bit is a binary digit which is added to ensure that the number of bits with value of one in a given set of bits is always even or odd. By using part of the capacity of the RAID for storing parity bits in a clever way, single disk failures can happen without data-loss since the missing bit can be recalculated using the parity bit. I/O transactions are movements of data to or from a RAID device or its members. Each I/O transaction consists of one or more blocks of data. A single disc can handle a maximum random number of transactions per second, since the mechanism has a seek time before data can be read or written. Depending on the configuration of the RAID, a single I/O transaction to the RAID can trigger multiple I/O transactions to its members. This affects the performance of the RAID device in terms of maximum number of I/O transactions and data transfer rate. Data transfer rate is the amount of data a single disk or RAID device can handle per second. This value usually varies for read and write actions, random or sequential access etc.

RAID is a method by which information is spread across several disks, using techniques such as disk striping (RAID Level 0), disk mirroring (RAID level 1), and striping with distributed parity (RAID Level 5), 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 must 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 transfer speed
- enhanced number of transactions per second
- increased single block device capacity
- greater efficiency in recovering from a single 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. For the LPIC-2 exam the following are relevant:

- 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 sum of the capacity of the member disks. As a result, RAID 0 is primarily used in applications that require high performance and are able to tolerate lower reliability.

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 the smallest member disk.

Level 4 RAID level 4 uses parity concentrated on a single disk drive to protect data. It is 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. Because the widespread use of modern CPUs and software RAID that is not really an issue anymore. 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. The capacity of the array is equal to the total capacity of all member disks, minus the capacity of one member disk. Upon failure of a single member disk, subsequent reads can be calculated from the distributed parity such that no data is lost. RAID 5 requires at least three disks.

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. The difference with "striping" is that there is no performance gain for single process applications, mostly everything is written to one and the same disk. The disk(partition)s can have different sizes whereas "striping" requires them to be roughly the same size. If you have a larger number of mostly used disks in a linear RAID setup, multiple processes may benefit during reads as each process may access a different drive. 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 array(s) 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 SATA disks as well as SCSI disks. With today's fast CPUs, software RAID performance can excel in comparison with hardware RAID.

Software RAID allows you to dramatically increase Linux disk I/O performance and reliability without having to buy 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. Also, the implementation and setup of the software RAID solution can significantly influence performance.

The concept behind software RAID is simple - it allows you to combine two (three, at least, for RAID5) 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 be used like any other partition.

Recognizing RAID on your Linux system

Detecting hardware raid on a Linux system can be tricky and there is not one sure way to do this. Since hardware RAID tries to present itself to the operating system as a single block device, it often shows up as a single SCSI disc when querying the system. Often special vendor software or physical access to the equipment is required to adequately detect and identify hardware RAID equipment. Software raid can be easily identified by the name of the block device (/dev/mdn) and its major number 9.

Configuring RAID (using mdadm)

Configuring software RAID using **mdadm** (Multiple Devices Admin) requires only that the md driver be configured into the kernel, or loaded as a kernel module. The optional `mdadm.conf` file may be used to direct **mdadm** in the simplification of common tasks, such as defining multiple arrays, and defining the devices used by them. The `mdadm.conf` has a number of possible options, described later in this document, but generally, the file details arrays and devices. It should be noted that, although not required, the `mdadm.conf` greatly reduces the burden on administrators to "remember" the desired array configuration when launching. The **mdadm** is used (as its acronym suggests) to configure and manage multiple devices. *Multiple* is key here. In order for RAID to provide any kind of redundancy to logical disks, there must obviously be at the very least two physical block devices (three for RAID5) in the array to establish redundancy, ergo protection. Since **mdadm** manages multiple devices, its application is not limited solely to RAID implementations, but may be also be used to establish multi-pathing. **mdadm** has a number of modes, listed below

Assemble "Rebuilds" a pre existing array. Typically used when migrating arrays to new hosts, but more often used from system startu
Build Does not create array superblocks, and therefore does not destroy any pre existing data. May be useful when attempting to reco
Create Creates an array from scratch, using pre-existing block devices, and activates the array.
Grow Used to modify a existing array, for example adding, or removing devices. Capability is expected to be extended during the dev
Misc Used for performing various loosely bundled housekeeping tasks. Can be used to generate the initial mdadm.conf file for an exi

Linux: `/etc/mdadm.conf` 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. **mdadm** may also be used to establish multipathing, also over filesystem devices (since multipathing is established at the block level). However, as with establishing RAID over multiple filesystems on the same physical disk, multipathing on the same physical disk provides only the vague illusion of redundancy, and its use should probably be restricted to test purposes only or out of pure curiosity.

Follow these steps to set up software RAID in Linux:

1. Configure the RAID driver
2. Initialise the RAID drive
3. Check the replication using `/proc/mdstat`
4. Automate RAID activation after reboot
5. 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 A driver file for each independant array will be automatically created when **mdadm** creates the array, and will follow the convention `/dev/md[n]` for each increment. It may also be manually created using **mknod** and building a block device file, with a major number 9 (md device driver found in `/proc/devices`).

Initialize RAID drive

Here is an example of the sequence of commands needed to create and format a RAID drive (mdadm):

1. Prepare the partition for auto RAID detection using **fdisk**

   In order for a partition to be automatically recognised as part of a RAID set, it must first have its partition type set to "fd". This may be achieved by using the **fdisk** command menu, and using the **t** option to change the setting. Available settings may be listed by using the **l** menu option. The description may vary accross implementations, but should clearly show the type to be a Linux auto raid. Once set, the settings *must* be saved back to the partition table using the **w** option.

   In working practice, it may be that a physical disk containing the chosen partition for use in a RAID set also contains partitions for local filesystems which are not intended for inclusion within the RAID set. In order for **fdisk** to write back the changed partition table, all of the partitions on the physical disk must not be in use. For this reason, RAID build planning should take into account factors which may not allow the action to be performed truly online (i.e will require downtime).

2. create the array raidset using **mdadm**

   To create an array for the first time, we need to have identified the partitions that will be used to form the RAIDset, and verify with **fdisk -l** that the fd partition type has been set. Once this has been achieved, the array may be created as follows. **mdadm** -C /dev/md0 -l raid5 -n 3 /dev/partition1 /dev/partition2 /dev/partition3

   This would create, and activate a raid5 array called md0,containing three devices, named /dev/partition1 /dev/partition2, and /dev/partiton3. Once created and running, the status of the array may be checked by **cat**'ing `/proc/mdstat`.

3. Create filesystems on the newly created raidset

   The newly created RAID set may then be addressed as a single device using the `/dev/md0` device file, and formatted and mounted as normal. For example: **mkfs.ext3** `/dev/md0`.

4. create the `/etc/mdadm.conf` using the **mdadm** command.

   Creating the `/etc/mdadm.conf` file is pleasantly simple, requiring only that **mdadm** be called with the `--scan`, `--verbose`, and `--detail` options, and its standard output redirected. It may therefore also be used as a handy tool for determining any changes on the array simply by **diff**'ing the current and stored output. Create as follows:

   mdadm --detail --scan --verbose > `/etc/mdadm.conf`

5. Create mount points and edit `/etc/fstab`

   Care must be taken that any recycled partitions used in the RAID set be removed from the `/etc/fstab` if they still exist.

6. Mount filesystems using **mount**

   If the array has been created online, and a reboot has not been executed, then the file systems will need to be manually mounted, either manually using the **mount** command, or simply using **mount -a**

Check the replication using /proc/mdstat The `/proc/mdstat` file shows the state of the kernels RAID/md driver.

Automate RAID activation after reboot

Run **mdadm `--assemble`** in one of the startup files (e.g. `/etc/rc.d/rc.sysinit` or `/etc/init.d/rcS`) When called with the `-s` option (scan) to **mdadm `--assemble`**, instructs **mdadm** to use the `/etc/mdadm.conf` if it exists, and otherwise to fallback to `/proc/mdstat` for missing information. A typical system startup entry could be for example, **mdadm `--assemble -s`**.

mount the filesystem on the RAID drive Edit `/etc/fstab` to automatically mount the filesystem on the RAID drive. The entry in `/etc/fstab` is identical to normal block devices containing file systems.

Manual RAID activation and mounting Run **mdadm --assemble** for each RAID block device you want to start manually. After starting the RAID device you can mount any filesystem present on the device using **mount** with the appropriate options.

Configuring RAID (alternative)

Instead of managing RAID via **mdadm**, it can be configured via `/etc/raidtab` and controlled via the **mkraid**, **raidstart**, and **raidstop** utilities.

Here's an example `/etc/raidtab`:

```
#
# Sample /etc/raidtab configuration file
#
raiddev /dev/md0
  raid-level              0
  nr-raid-disks           2
  persistent-superblock   0
  chunk-size              8

  device                  /dev/hda1
  raid-disk               0
  device                  /dev/hdb1
  raid-disk               1
```

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

  device                  /dev/sda1
  raid-disk               0
  device                  /dev/sdb1
  raid-disk               1
  device                  /dev/sdc1
  raid-disk               2
  device                  /dev/sdd1
  spare-disk              0
```

The **mkraid**, **raidstart**, and **raidstop** utilities all use this file as input and respectively configure, activate (start) and unconfigure (stop) RAID devices. All of these tools work similiarly. If -a (or --all) is specified, the specified operation is performed on all of the RAID devices mentioned in the configuration file. Otherwise, one or more RAID devices must be specified on the command line.

Refer to the manual pages for raidtab(8), mdstart(8) and raidstart, raidstop(8) for details.

Adjusting Storage Device Access (204.2)

Candidates should be able to configure kernel options to support various drives. This objective includes software tools to view and modify hard disk settings including iSCSI devices.

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

Key Knowledge Areas

- Tools and utilities to configure DMA for IDE devices including ATAPI and SATA
- Tools and utilities to manipulate or analyse system resources (e.g. interrupts)
- Awareness of **sdparm** command and its uses
- Tools and utilities for iSCSI
- SSD and NVMe configuration and awareness of SAN

Terms and Utilities

- **hdparm**
- **sdparm**
- **tune2fs**
- **sysctl**
- **iscsiadm, scsi_id, iscsid** and `iscsid.conf`
- `/dev/hd*` and `/dev/sd*`
- WWID, WWN, LUN numbers

Configuring disks

Configuring iSCSI

iSCSI is an abbreviation of Internet Small Computer System Interface. iSCSI is simply a networked implementation of the well-known standardized SCSI protocol. SCSI defines a model, cabling, electrical signals and commands to use to access various devices. iSCSI uses just the model and the commands. SCSI (and hence: iSCSI) can be used to access all kinds of devices and though mostly used for disks and tape-units has been used to access USB storage, printers and flatbed scanners too, just to name a few.

The iSCSI setup is similar to the SCSI setup as it consists of a client and a server. In between is an IP based network, which can be a LAN, WAN or the Internet. The client issues low level SCSI commands as usual (e.g. **read**, **write**, **seek** or **erase**). The commands can be encrypted by the client and are then encapsulated and sent over the network. The server receives and possibly decrypts the commands and executes the commands. Unlike traditional Fibre Channel, which requires special-purpose cabling, iSCSI can be run over long distances using existing network infrastructure.

Clients are refered to as "initiators". The commands are referred to as "CDBs" (Command Descriptor Blocks) and the server storage devices are known as "targets".

Definitions:

iSCSI target
iSCSI initiator
The exported storage entity is the *target* and the importing entity is the *initiator*.

iSCSI Initiator

On Linux hosts that act as a client (initiator) to a iSCSI server (target) you will need to install the client software. You will also need to edit the configuration file so you can discover the desired target(s). On Red Hat Linux and its derivatives the configuration file is `/etc/iscsi/iscsid.conf`. Edit the file so it points to the proper server (target) and contains a set of valid credentials - see example file at the bottom of this segment.

After configuration, ensure that the iSCSI service runs using the following command:

```
/etc/init.d/iscsi status
```

If needed start the iSCSI service using the following command:

```
/etc/init.d/iscsi start
```

Use the **iscsiadm** command to discover the targets. Where our example shows questionmarks, substitute the IP address of the target:

```
iscsiadm -m discovery -t sendtargets -p ???.???.???.???
```

After successfully discovering targets you can log in into the volumes that were discovered. This can be done with the **iscsiadm** tool, which can also be used to log out of a volume. A fast way to add newly discovered volumes is to restart the iscsi service:

```
/etc/init.d/iscsi restart
```

Tip
When adding an iscsi mount point to the `fstab` file use the `_netdev` option:

```
/dev/sdb1 /mnt/iscsi ext3(4) _netdev 0 0
```

The `_netdev` option ensures that the network is up before trying to mount the device.

Example `/etc/iscsi/iscsid.config`:

```
################
# iSNS settings
################
# Address of iSNS server
isns.address = ***.***.***.***
isns.port = 3260

# *************
# CHAP Settings
# *************

# To enable CHAP authentication set node.session.auth.authmethod
# to CHAP. The default is None.
node.session.auth.authmethod = CHAP

# To set a CHAP username and password for initiator
# authentication by the target(s), uncomment the following lines:
node.session.auth.username = ********
node.session.auth.password = ********

# To enable CHAP authentication for a discovery session to the target
# set discovery.sendtargets.auth.authmethod to CHAP. The default is None.
discovery.sendtargets.auth.authmethod = CHAP

# To set a discovery session CHAP username and password for the initiator
# authentication by the target(s), uncomment the following lines:
discovery.sendtargets.auth.username = ********
discovery.sendtargets.auth.password = ********
```

iSCSI Target

There are a number of ways to set up a target. The SCSI Target Framework (STGT/TGT) was the standard before linux 2.6.38. The current standard is the LIO target.

Another well-known implementation was the iSCSI Enterprise Target (IET) and its successor the SCSI Target Subsystem (SCST). IET was a candidate for kernel inclusion too, but LIO was the final choice. LIO (an abbreviation of linux-iscsi.org) now is the standard open-source SCSI Target for shared data storage in Linux. It supports all prevalent storage fabrics, including Fibre Channel (QLogic), FCoE, iEEE 1394, iSCSI, iSER (Mellanox InfiniBand), SRP (Mellanox InfiniBand), USB, vHost, etc.

Adding a new target (on the target host)

To add a new iSCSI target edit the `/etc/tgt/targets.conf` configuration file. This file contains many examples of different configuration options that have been commented out. A basic target may be defined as:

```
<target iqn.2008-09.com.example:server.target1>
    backing-store /srv/images/iscsi-share.img
    direct-store /dev/sdd
</target>
```

This example defines a single target with two LUNs. LUNs are described with either the backing-store or direct-store directives where backing-store refers to either a file or a block device, and direct-store refers to local SCSI devices. Device parameters, such as serial numbers and vendor names, will be passed through to the new iSCSI LUN.

The target service is aptly named **tgtd**. To start it run:

```
# service tgtd start
```

To stop the `tgtd` service, run:

```
# service tgtd stop
```

If there are open connections, use:

```
# service tgtd force-stop
Warning Using this command will terminate all target arrays.
```

WWID

Any connected SCSI device is assigned a unique device name. The device name begins with `/dev/sd`, for example `/dev/sdd`, `/dev/sda`. Additionally, each SCSI device has a unique World Wide Identifier (WWID). The identifier can be obtained from the device itself by issuing the **inquiry** command. On Linux systems you can find the WWIDs in the `/dev/disk/by-id/` directory, in which you will find symbolic links whose names contain the WWID and which point to the assigned device name. There are two types of WWIDs, known as 'page 0x83' and 'page 0x80' and any SCSI device has one of these.

For example, a device with a page 0x83 identifier would have:

```
scsi-3600508b400105e210000900000490000 -> ../../sda
```

Or, a device with a page 0x80 identifier would have:

```
scsi-SSEAGATE_ST373453LW_3HW1RHM6 -> ../../sda
```

Red Hat Enterprise Linux automatically maintains the proper mapping from the WWID-based device name to a current `/dev/sd` name on that system. Applications can use the `/dev/disk/by-id/` name to reference the data on the disk, even if the path to the device changes, and even when accessing the device from different systems. This also allows detection and access to plugable devices, say a photocamera. If it is plugged in, the system will sent a **inquiry** command and will create the proper symbolic link, hence allowing you to access the camera under the same name.

It is possible and feasible to have more than one path to a device, for example to offer a fail-over or fall-back option and/or to increase performance. Such "paths" might be implemented as additional network paths (preferably using their own network card) or over fibre (preferably using their own HBAs) etc. If there are multiple paths from a system to a device, a special kernel-module and associated daemons and other software will use the WWID to detect the paths. A single "pseudo-device" in `/dev/mapper/`*`wwid`* will be created by them, such as `/dev/mapper/3600508b400105df70000e00000ac0000`, which will give access to the device, regardless which path(s) to it there are in use, as long as at least one of them can be used ('is active').

The command **multipath `-l`** shows the mapping to the non-persistent identifiers: Host:Channel:Target:LUN, `/dev/sd` name, and the major:minor number:

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=0][active]
\_ 5:0:1:1 sdc 8:32  [active][undef]
\_ 6:0:1:1 sdg 8:96  [active][undef]
\_ round-robin 0 [prio=0][enabled]
\_ 5:0:0:1 sdb 8:16  [active][undef]
\_ 6:0:0:1 sdf 8:80  [active][undef]
```

Device-mapper-multipath automatically maintains the proper mapping of each WWID-based device name to its corresponding `/dev/sd` name on the system. These names are persistent a cross path changes, and they are consistent when accessing the device from different systems. When the user_friendly_names feature (of device-mapper-multipath) is used, the WWID is mapped to a name of the form `/dev/mapper/`*`mpathn`*. By default, this mapping is maintained in the file `/etc/multipath/bindings`. These **mpath** names are persistent as long as the file is maintained.

Note
Important If you use user_friendly_names, then additional steps are required to obtain consistent names in a cluster. Refer to the Consistent Multipath Device Names in a Cluster section in the Using DM Multipath Configuration and Administration book.

LUN

Another way to address a SCSI device is by using its LUN, the Logical unit number. The LUN is a three bit number (0..7) which indicates a device within a target. Any target should at least have a device with LUN 0, which can be used like any device, but is also capable of providing information about the other LUNs in the device (if any). If there is just one device in a target that device has LUN 0 as any target should have a LUN 0.

Say you bought a SCSI device that contained three disks (all connected over the same SCSI cable). To find out how many disks there are in your target (or tape units etc.) you would sent an inquiry to LUN 0. The device that has LUN 0 is designed to be able to tell you the configuration and might report that it knows of other devices. To address these the client uses the LUNs as specified by LUN 0.

Because SCSI disks were often refered to by their LUN, the habit stuck and nowadays the term is also used to refer to a logical disk in a SAN. But there can be many more LUNs in a SAN than the humble 7 that are used in older configurations. Also, such a SAN-LUN may well consist of series of disks, which might possibly be configured as a RAID, possibly subdivided in volumes etc. - to the client, it is just a disk with a LUN.

Configuring device name persistence (aka LUN persistence)

Linux device names may change upon boot. This can lead to confusion and damage to data. Therefore, there are a number of techniques that allow persistent names for devices. This section covers how to implement device name persistence in guests and on the host machine with and without multipath. Persistence simply means that you will have the same name for the device, regardles its access paths. So, your camera or disk is always recognised as such and will be issued the same device name, regardless where you plug it in.

Implementing device name persistence without multipath

If your system is not using multipath, you can use **udev** to implement LUN persistence. **udev** is a device manager that listens to an interface with the kernel. If a new device is plugged in or a device is being detached, **udev** will receive a signal from the kernel and, using a set of rules, will perform actions, for example assign an additional device name to the device.

Before implementing LUN persistence in your system you will first need to acquire the proper WWIDs from the devices you want to control. This can be done by using the `scsi_id` program. After you required the WWIDs of your devices, you will need to write a rule so **udev** can assign a persistent name to it.

A minor hurdle has to be taken first. The **scsi_id** program will not list any device by default and will only list devices it has been told to see ('whitelisted' devices). To whitelist a device you will need to add its WWID to the **scsi_id** configuration file, which is `/etc/scsi_id.conf`. That's safe, but a nuisance if you want to discover WWIDs of new devices. Hence you can specify the `-g` option to allow detection of any disk that has not been listed yet. If you feel that this should be the default behaviour, you may consider editing `/etc/scsi_id.conf`. If there is a line that says:

```
options=-b
```

delete it or comment it out. Then add a line that says:

```
options=-g
```

If this has been configured, next time a program uses the **scsi_id** command, it will add the `-g` option (and any other options you might have specified in the configuration file) and hence allow extracting the WWIDs from new devices. The `-g` should be used either by default from the configuration file or be set manually in any **udev** rule that uses the **scsi_id** command, or else your newly plugged in device won't be recognised.

You then will need to find out what device file your new device was allocated by the kernel. This is routinely done by using the command **dmesg** after you plugged in the new device. Typically, you will see lines similar to these:

```
usb 1-5: new high speed USB device using ehci_hcd and address 25
usb 1-5: configuration #1 chosen from 1 choice
scsi7 : SCSI emulation for USB Mass Storage devices
usb-storage: device found at 25
usb-storage: waiting for device to settle before scanning
```

```
  Vendor: USB       Model: Flash Disk        Rev: 8.07
  Type:   Direct-Access                      ANSI SCSI revision: 04
SCSI device sdc: 7886848 512-byte hdwr sectors (4038 MB)
```

As you see, we plugged in a 4Gb Flash Disk that was assigned the device name 'sdc'. This device is now available as `/sys/block/sdc` in the `/sys` pseudo-filesystem, and also as a devicefile `/dev/sdc`.

To determine the device WWID, use the **scsi_id** command:

```
# scsi_id -g -u -s /block/sdc
3200049454505080f
```

As you see, we used the `-g` forcibly. The `-s` option specifies that you want to use the device name relative to the root of the **sysfs** filesystem (`/sys`). Newer versions of scsi_id (e.g. those in RHEL6) do not allow use of **sysfs** names anymore, so should use the device name then:

```
/sbin/scsi_id -g -u -d /dev/sdc
3200049454505080f
```

The long string of characters in the output is the WWID. The WWID does not change when you add a new device to your system. Acquire the WWID for each device in order to create rules for the devices. To create new device rules, edit the `20-names.rules` file in the `/etc/udev/rules.d` directory. The device naming rules follow this format:

```
KERNEL="sd<?>",  BUS="scsi",  PROGRAM="<scsi_id -options..>", RESULT="<WWID>", NAME="< ↩
    devicename>"
```

So, when you plug in a new device the kernel will signal **udev** it found a new SCSCI device. the PROGRAM will be executed and the RESULT should match. If so, the NAME will be used for the device file.

An example would be:

```
KERNEL="sd*",  BUS="scsi",  PROGRAM="/sbin/scsi_id -g -u -d /dev/$parent", RESULT ↩
    ="3200049454505080f", NAME="bookone"
```

Or on RHEL6 systems:

```
KERNEL="sd*",  BUS="scsi",  PROGRAM="/sbin/scsi_id -g -u -s /block/$parent", RESULT ↩
    ="3200049454505080f", NAME="bookone"
```

When the kernel allocates a new device which name matches the KERNEL pattern, the PROGRAM is executed and when the RESULT is found the NAME is created. Note the `-g` option that we have included so we can actually see the device, though it is not whitelisted.

The **udevd** daemon is typically started by the execution of a local startup script, normally done by the **init** process and initiated by a line in the `/etc/inittab` file. On older systems you would ensure that this line was in `/etc/rc.local`:

```
/sbin/start_udev
```

On some systems, e.g. RHEL6 the line should be in `/etc/rc.sysinit`.

Implementing device name persistence with multipath

You can implement device name persistence by using the **multipath** drivers and software. Of course, normally you would only use this if you have more than one path to your device. But you can actually define a **multipath** with only has one active path, and it will work just like any other disk. Just define an alias and the name will be persistent across boots.

The **multipath** software consists of a daemon, that guards the paths to the various devices. It executes the external command **multipath** on failure. The **multipath** command in turn will signal the daemon after it has reconfigured paths.

To implement LUN persistence in a multipath environment, you can define an alias name for the **multipath** device. Edit the device aliases in the configuration file of the **multipath** daemon, `/etc/multipath.conf`:

```
multipaths {
    multipath  {
        wwid       3600a0b80001327510000015427b625e
        alias      backupdisk
    }
}
```

This defines a device that has a persistent device file `/dev/mpath/backupdisk`. That will grant access to the device - whatever its name - that has WWID 600a0b80001327510000015427b625e. The alias names are persistent across reboots.

Physical installation

To install a new disk in your system, you will need to physically install the hard drive and configure it in your computer's BIOS. Linux will detect the new drive automatically in most cases. You could type **dmesg | more** to find out the name and device file of the drive. As an example: the second IDE drive in your system will be named `/dev/hdb`. We will assume that the new drive is `/dev/hdb`.

On older kernels IDE devicenames match the pattern `/dev/hd[a-d]`, where a is used for the primary master IDE device, b is the primary slave, c the secondary master and d is the secondary slave IDE device. For PATA/SATA drives the Linux SCSI driver is used as the PATA/SATA architecture is similar to that of SCSCI. Newer devices hence have device files that match the pattern `/dev/sd*`.

After installation of your disk and it being recognised by the kernel you can choose to (re)partition your new disk. As **`root`** in a shell type:

```
fdisk /dev/hdb
```

This will take you to a prompt

```
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. (Creating Filesystems).

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

Modern ssd's (Solid State Disks) might profit from some optimisation. Ssd's are very fast compared to standard harddisks with platters but they have a bigger chance of suffering from bad blocks after many rewrites to blocks. An ssd can use TRIM you discard those blocks efficiently. **noatime** or **relatime** can also be used to reduce the amount of writes to disk. This will tell the filesystem not to keep track of last accessed times, but only last modified times. In `/etc/fstab` it would look like:

```
/dev/hdb1         /new      ext4     discard,noatime        1       1
```

NVM Express (NVMe) is a specification for accessing SSDs attached through the PCI Express bus. The Linux NVMe driver is included in kernel version 3.3 and higher. NVMe devices can be found under `/dev/nvme*`. NVMe devices are a family of ssd's thatshould not be issued discards. Discards are usualy disabled on setups that use ext4 and LVM, but other filesystems might need discards to be disabled explicitly.

```
/dev/nvme0n1p1  /new     ext4 defaults 0 0
```

Moving your `/tmp` partition to memory (tmpfs) is another way to reduce disk writes. This is advisable if your system has enough memory. Add the following entry to `/etc/fstab`:

```
none             /tmp/                   tmpfs   size=10%                0       0
```

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`.

Using tune2fs

When a kernel interacts with ext2 or ext3 filesystems the **tune2fs** command can be used to configure the interaction. Noteworthy options of this command are:

-e error_behaviour This option defines how the kernel should react in case of errors on the filesystem to which this command is applied. Possible behaviours are:

- **continue** Ignore the error on the filesystem. Report the read or write error to the application that tries to access that part of the filesystem.
- **remount-ro** Remount the filesystem as read-only. This secures data that is already on disc. Applications that try to write to this filesystem will result in error.
- **panic** This causes a kernel panic and the system will halt.

-m reserved_block_percentage Full filesystems have very bad performance due to fragmentation. To prevent regular users from filling up the whole filesystem a percentage of the blocks will be reserved. This reserved part cannot be used by regular users. Only the root user can make use this part of the filesystem.

The default percentage of the reserved blocks is 5%. With disks becoming ever larger, setting aside 5% would result in a large number of reserved blocks. For large disks 1% reserved blocks should be sufficient for most filesystems.

-O [ˆ]mount_option This sets the mount options of the filesystem. Command line options or options in `/etc/fstab` take precedence. The ˆ-sign clears the specified option.

This option is supported by kernel 2.4.20 and above.

-s [0|1] Enable or disable the sparse superblock feature. Enabling this feature on large filesystems saves space, because less superblocks are used.

After enabling or disabling this feature the filesystem must be made valid again by running the **e2fsck** command.

Using hdparm

The **hdparm** command is used to set/get SATA/IDE device parameters. It provides a command line interface to various kernel interfaces supported by the Linux SATA/PATA/SAS "libata" subsystem and the older IDE driver subsystem. Newer (from 2008) USB disk enclosures now support "SAT" (SCSI-ATA Command Translation) so they might also work with **hdparm**.

The syntax for this command is:

```
hdparm [options] [device]
```

Frequently used options are:

-a Get or set the sector count for filesystem read-ahead.

-d [0|1] Get or set the `using_dma` flag.

-g Display drive geometry.

-i Display identification information.

-r [0|1] Get (1) or set (0) the read-only flag.

-t Perform device read for benchmarking.

-T Perform device cache read for benchmarking.

-v Display all settings, except -i.

See also **man hdparm** for more information.

Using sdparm

The **sdparm** command is used to access SCSI mode pages, read VPD (Vital Product Data) pages and send simple SCSI commands.

The utility fetches and can change SCSI device mode pages. Inquiry data including Vital Product Data pages can also be displayed. Commands associated with starting and stopping the medium; loading and unloading the medium; and other housekeeping functionality may also be issued by this utility.

Configuring kernel options

In Section 1.1 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 use the **echo** command to set parameters, in the format:

```
# echo value > /proc/kernel/parameter
```

e.g. to activate IP-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
```

On multicore systems you will see multiple CPU-columns, e.g. CPU0..CPU3 for a four core system. The interrupts are delivered to core 0 by default. This can create a performance bottleneck, especially on networked storage systems. As CPU0 gets swamped by IRQ's the system feels sluggish, even when the other cores have almost nothing to do. Newer kernels try to load balance the interrupts. But you can override the defaults and choose which core will handle which interrupt. The procedure is listed in the next paragraph.

Each interrupt listed in the `/proc/interrupts` file has a subdirectory in the `/proc/irq/` tree. So, interrupt 12 corresponds to the directory `/proc/irq/12`. In that directory the file `smp_affinity` contains data that describes what core handles the interrupt. The file contains a bitmask that has one bit per core. By setting the proper bit the APIC chip will deliver the interrupt to the corresponding core, e.g. 2 selects CPU1, 4 selects CPU2, 8-3, 16-4.. etc. To set for example CPU5 to receive all interrupts #13, do:

```
# echo 32 >/proc/irq/13/smp_affinity
```

By carefully selecting which interrupts go to which processor you can dramatically influence the performance of a busy system.

Using sysctl

Kernel tuning can be automated by putting the **sysctl** commands in the startup scripts (e.g. `/etc/sysctl.d/99-sysctl.conf`. The **sysctl** command is used to modify kernel parameters at runtime. The parameters that can be modified by this command are listed in the `/proc/sys` directory tree. Procfs is required for **sysctl** support under Linux. **sysctl** can be used to read as well as write kernel parameters.

Frequently used options of the **sysctl** command are:

-a, -A Display all values currently available.

-e Use this option to ignore errors about unknown keys.

-n Use this option to disable printing of the key name when printing values.

-p Load sysctl settings from the file specified, or /etc/sysctl.conf if none given. Specifying – as filename means reading data from standard input.

-w Use this option to change a sysctl setting.

Logical Volume Manager (204.3)

Candidates should be able to create and remove logical volumes, volume groups, and physical volumes. This objective includes snapshots and resizing logical volumes.

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

Key Knowledge Areas

- Tools in the LVM suite
- Resizing, renaming, creating, and removing logical volumes, volume groups, and physical volumes
- Creating and maintaining snapshots
- Activating volume groups

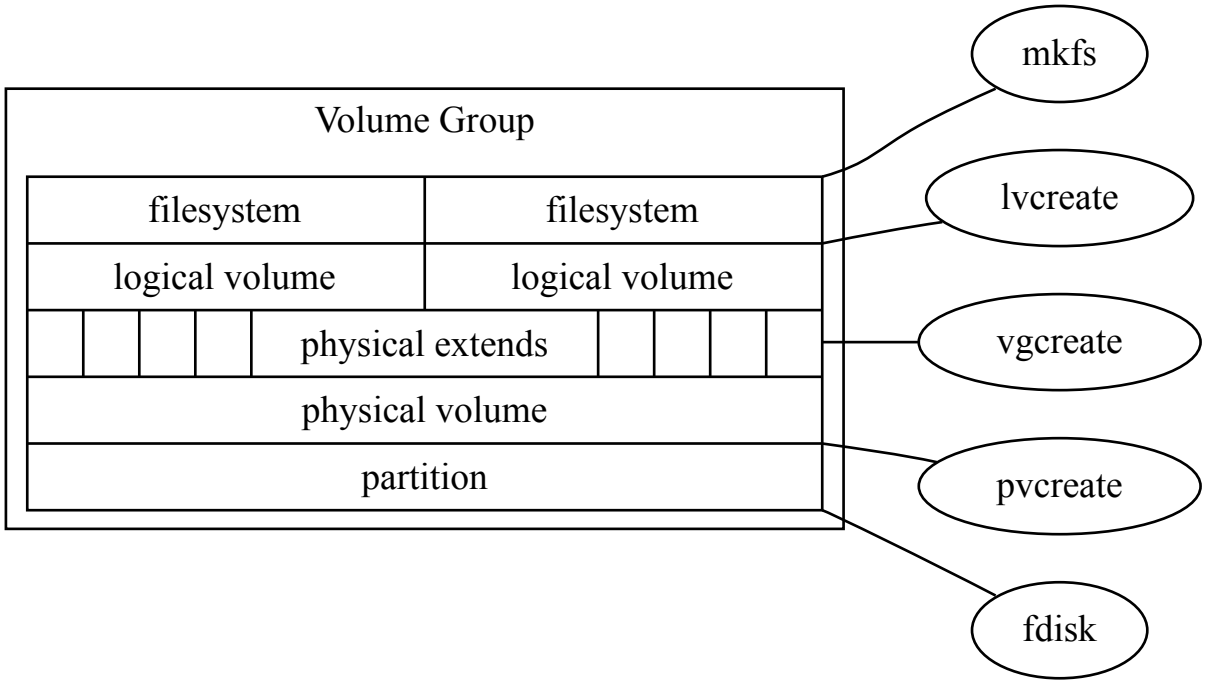
Terms and Utilities

- **/sbin/pv***
- **/sbin/vg***
- **/sbin/lv***
- **mount**
- `/dev/mapper`
- `lvm.conf`

Configuring Logical Volume Management

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

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



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 -t xfs /dev/volume01/lvol0
```

The resulting filesystem can be mounted as usual:

```
# mount /dev/volume01/lvol0 /mnt
```

Modifying logical volumes, volume groups and physical volumes

A logical volume can be modified in order to create more space for the filesystem that is on top of this logical volume. Assuming that there is enough space in the volume group a logical volume can be increased in size with the command:

```
# lvextend -L +50M /dev/volume01/lvol0
```

After the logical volume has been increased the filesystem on top of the logical volume still has the same size. To use the extra space of the logical volume the filesystem needs to be resized. This can be done by the command:

```
# xfs_growfs /dev/volume01/lvol0
```

For an ext2/ext3/ext4 file system use the command **resize2fs**. Note that for resizing the filesystem it must not be mounted.

In the previous example it was assumed that there was enough free space in the volume group. If this is not the case, extra disk space can be added to the volume group in a similar way. To do so use the command:

```
# vgextend volume01 /dev/hda6
```

First device hda6 has to be converted into a physical volume with the command:

```
# pvcreate /dev/hda6
```

LVM Snapshots

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. LVM snapshots use a strategy called "copy on write". This means that the snapshot logical volume only saves data blocks from the original logical volume that are changed in the original logical volume. To do so the logical volume manager first reads the (unchanged) data block on the original and than writes the data block to the snapshot. On filesystems with many changes (e.g. databases) this can lead to performance issues.

The **-s** option in the **lvcreate** command specifies that the newly created logical volume is a snapshot.

```
# lvcreate -L 50M -s -n snapshot0 /dev/volume01/lvol0
```

This will create a logical volume `/dev/volume01/snapshot0`, which then can be used, among others, for backup purposes. The advantage of the snapshot is that the data is consistent, i.e. the data doesn't change during backup.

After the backup is finished, the snapshot has to be removed otherwise the performance issues mentioned earlier will start to come into play. To remove the snapshot use:

```
# lvremove /dev/volume01/snapshot0
```

LVM commands

This section gives an overview of most lvm related commands. The manual pages describe the commands more in detail.

pvchange Change attributes of a physical volume

pvck Check physical volume metadata

pvcreate Initialize a disk or partition for use by LVM

pvdisplay Display attributes of a physical volume

pvmove Move physical extents

pvremove Remove a physical volume

pvs Report information about physical volumes

pvscan Scan all disk for physical volumes

lvchange Change attributes for a logical volume

lvcreate Create a logical volume in an existing volume group

lvdisplay Display attributes of a logical volumes

lvextend Extend the size of a logical volume

lvmdiskscan Scan for all devices visible to LVM2

lvreduce Reduce the size of a logical volume

lvremove Remove a logical volume

lvrename Rename a logical volume

lvresize Resize a logical volume

lvs Report information about logical volumes

lvscan Scan (all) disks for logical volumes

vgcfgbackup Backup volume group descriptor area

vgchange Change attributes of a volume group

vgck Check volume group metadata

vgconvert Convert volume group metadata

vgcreate Create a volume group

vgdisplay Display attributes of volume groups

vgextend Add physical volumes to a volume group

vgexport Make volume groups unknown to the system

vgimport Make exported volume groups known to the system

vgmerge Merge two volume groups

vgmknodes Recreate volume group directory and logical volume special files

vgreduce Reduce a volume group

vgremove Remove a volume group

vgrename Rename a volume group

vgs Report information about volume groups

vgscan Scan all disk for volume groups and rebuild caches

vgsplit Split a volume group into two

Device mapper

The device mapper is a kernel driver that provides a framework for volume management. It provides a generic way of creating mapped devices, which may be used as logical volumes. It does not specifically know about volume groups or metadata formats.

LVM logical volumes are activated using the device mapper. Each logical volume is translated into a mapped device. Each segment translates into a line in the mapping table that describes the device. The device mapper supports a variety of mapping targets, including linear mapping, striped mapping, and error mapping.

lvm.conf

At system startup `lvm.conf` configuration is loaded. The default directory is `/etc/lvm`. But it can be set with the environment variable LVM_SYSTEM_DIR. In the `lvm.conf` file you can specify additional configuration to load. To display the current settings, you can execute the lvm dumpconfig command.

The **vgscan** command scans for block devices with lvm metadata on it. These physical volumes are stored in the lvm cache. This command uses the `lvm.conf` file to determine that. When for instance you use multipath devices, the metadata is the same on all the disks, on the different paths. Which will cause a lot of duplicates when you run **pvs**. Also when you want to boot of a multipath device, it can boot of a single path. You can use the filter option in the devices section for this in the `lvm.conf`, to skip the other devices and only look for multipath devices. The filter option uses regular expression to accept or reject block devices, like in below examples. Add all discovered devices:

```
filter = [ "a/.*/" ]
```

Filter to remove cdrom devices:

```
filter = [ "r|/dev/cdrom|" ]
```

Filter only device mapper names on a multipath device:

```
filter = [ "a|/dev/disk/by-id/dm-uuid-.*-mpath-.*|", "r|.*|" ]
```

This filter just adds partition 8 on the first IDE Harddrive and removes all other block devices.

```
filter = [ "a|^/dev/hda8$|", "r/.*/" ]
```

Other usefull and commonly used option in the devices sections in the `lvm.conf` are: preferred_names The earliest pattern is used in any output like **pvs**. So in our example of multipath devices we want.

```
preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath", "^/dev/[hs]d" ]
```

types If you boot from a local device and the rest are multipath devices, you need to change the types option, next to the filter option with this example. I tells LVM the acceptable block devices (device-mapper) and the allowed partitions (253)

```
types = [ "device-mapper", 253 ]
```

For other sections with options check: **man lvm.conf**

Questions and answers

Advanced Storage Device Administration

1. *What is RAID?*

   RAID is a method by which information is spread across several disks, using techniques to achieve redundancy, lower latency and/or higher bandwidth for reading and/or writing to disk, and maximize recoverability from hard-disk crashes. What is RAID? [102]

2. *Is it true that using linear RAID increases the level of redundancy?*

   No, in fact it decreases reliability -- if any one member drive fails, the entire array cannot be used. Linear RAID

3. *How would you automate RAID activation after reboot using `mdadm.conf`, if it exists?*

   Run **mdadm --assemble -s** in one of the startup files. Automate RAID activation after reboot

4. *What is the objective of using Logical Volume Management (LVM)?*

   It enables you to concatenate several physical volumes (hard disk, etc) to a so-called volume group, forming a storage pool. Configuring Logical Volume Management [116]

5. *What is a Logical Volume (LV)?*

   A Logical Volume is created based on a number of Physical Extents (PE) taken from a Volume Group (VG) on top of which a filesystem can be built. Logical Volume (LV) [116]

Chapter 5

Networking Configuration (205)

This topic has a weight of 11 points and contains the following objectives:

Objective 205.1; Basic Networking Configuration (3 points) Candidates should be able to configure a network device to be able to connect to a local, wired or wireless, and a wide-area network. This objective includes being able to communicate between various subnets within a single network.

Objective 205.2; Advanced Network Configuration and Troubleshooting (4 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.

Objective 205.3; Troubleshooting Network Issues (4 points) Candidates should be able to identify and correct common network setup issues to include knowledge of locations for basic configuration files and commands.

Basic Networking Configuration (205.1)

Candidates should be able to configure a network device to be able to connect to a local, wired or wireless, and a wide-area network. This objective includes being able to communicate between various subnets within a single network.

Resources: Dawson00, Drake00.

Key Knowledge Areas

- Utilities to configure and manipulate ethernet network interfaces
- Configuring wireless networks

Terms and Utilities

- **/sbin/route**
- **/sbin/ifconfig**
- **/sbin/ip**
- **/usr/sbin/arp**
- **/sbin/iw**
- **/sbin/iwconfig**
- **/sbin/iwlist**

Configuring the network interface

Most network devices are supported by modern kernels. But you will need to configure your devices to fit into your network. They will need an IP address (IPv4, IPv6 or both), possibly a number of gateways/routers has to be made known to allow access to other networks and the default route needs to be set. These tasks are usually performed from the network-initialization script each time you boot the system. The basic tools for this process are **ifconfig** (where "if" stands for interface) and **route**.

The **ifconfig** command is still widely used. It configures an interface and makes it accessible to the kernel networking layer. An IP address, submask, broadcast address and various other parameters can be set. The tool can also be used to activate and de-activate the interface, also known as "bringing up" and "bringing down" the interface. An active interface will send and receive IP datagrams through the interface. The simplest way to invoke it is:

```
# 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

TCP/IP implementations include a virtual network interface that can be used to emulate network traffic between two processes on the same host. The loopback interface is not connected to any real network, it is implemented entirely within the operating system's networking software. Traffic sent to the loopback IP address (often the address 127.0.0.1 is used) is simply passed back up the network software stack as if it had been received from another device. The IPv6 address used is ::1, and commonly the name "localhost" is used as hostname. It is the first interface to be activated during boot:

```
# ifconfig lo 127.0.0.1
```

Occasionally, you will see the dummy hostname localhost being used instead of the IP address. This requires proper configuration of the `/etc/hosts` file:

```
# Sample /etc/hosts entry for localhost
127.0.0.1 localhost
```

To view the configuration of an interface simply invoke **ifconfig** with the interface name as sole 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
```

This example shows that the loopback interface has been assigned a netmask of 255.0.0.0 by default. 127.0.0.1 is a class A address.

These steps suffice to use networking applications on a stand-alone host. After adding these lines to your network initialization script [1] and ensuring its execution at boot time by rebooting your machine you can test the loopback interface. For instance, **telnet localhost** should establish a telnet connection to your host, giving you a **`login:`** prompt.

The loopback interface is often used as a test bed during development, but there are other applications. 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. Hence the loopback interface should always be configured, whether your machine is attached to a network or not.

[1] This is usually included with your distribution.

Ethernet Interfaces

Configuring an Ethernet interface is pretty much the same as the loopback interface - it just requires a few more parameters when you use 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** invocation 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.

Routing Through a Gateway

You do not need routing if your host is on a single Ethernet. Quite frequently however, networks are 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 the routing table telling the kernel all hosts on the other network can be reached through `romeo`. The appropriate invocation 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 want datagrams to any destination network to be handed to `romeo`. This can be accomplished by making it the default gateway:

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

Something that is frequently misunderstood is that only *ONE* default gateway can be configured. You may have many gateways, but only one can be the default.

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.

The ip command

In recent years many people have advocated the use of the newer command **/sbin/ip**. It too can be used to show or manipulate routing and network devices, and also can be used to configure or show policy routing and tunnels. However, the old tools **ifconfig** and **route** can be used too if that is more convenient. A use case for the **ip** would be to show the IP addresses used on the network interfaces in a more concise way compared to **ifconfig**:

```
# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: p8p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:90:f5:b6:91:d1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.123.181/24 brd 192.168.123.255 scope global p8p1
    inet6 fe80::290:f5ff:feb6:91d1/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 88:53:2e:02:df:14 brd ff:ff:ff:ff:ff:ff
4: vboxnet0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 0a:00:27:00:00:00 brd ff:ff:ff:ff:ff:ff
```

in contrast to **ifconfig**

```
# ifconfig -a
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4304 (4.2 KiB)  TX bytes:4304 (4.2 KiB)

p8p1      Link encap:Ethernet  HWaddr 00:90:F5:B6:91:D1
          inet addr:192.168.123.181  Bcast:192.168.123.255  Mask:255.255.255.0
          inet6 addr: fe80::290:f5ff:feb6:91d1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6653 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7609 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3843849 (3.6 MiB)  TX bytes:938833 (916.8 KiB)
          Interrupt:53

vboxnet0  Link encap:Ethernet  HWaddr 0A:00:27:00:00:00
          BROADCAST MULTICAST  MTU:1500  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 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

wlan0     Link encap:Ethernet  HWaddr 88:53:2E:02:DF:14
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:330 (330.0 b)  TX bytes:1962 (1.9 KiB)
```

An example of using **ip** as an alternative to **ifconfig** when configuring a network interface (p8p1):

```
# ifconfig p8p1 192.168.123.15 netmask 255.255.255.0 broadcast 192.168.123.255
```

can be replaced by:

```
       ip addr add 192.168.123.15/24 broadcast 192.168.123.255 dev p8p1
```

The **ip** can also be used as alternative for the **route** command:

```
# ip route add 192.168.123.254/24 dev p8p1
```

```
# ip route show
192.168.123.0/24 dev p8p1  proto kernel  scope link  src 192.168.123.181  metric 1
default via 192.168.123.254 dev p8p1  proto static
```

The output of the **route** command in this case would be:

```
# route
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.123.0   *               255.255.255.0   U     1      0        0 p8p1
default         192.168.123.254 0.0.0.0         UG    0      0        0 p8p1
```

As another example, to add a static route to network 192.168.1.0 over eth0, use:

```
# ip route add 192.168.1.0/24 dev eth0
```

For more information please read the manual pages of **ip**.

ARP, Address Resolution Protocol

In the ISO network model there are seven layers. The Internet Protocol is a layer 3 protocol and the NIC is a layer 2 device. In a local network (layer 2) devices know each other by the MAC (Media Access Control) address. In the IP network (layer 3) devices know each other by their IP address.

To allow transfer from data to and from layer 3 IP communication requires a protocol to map between layer 2 and layer 3. This protocol is known as ARP - the Address Resolution Protocol. ARP creates a mapping between an IP address and the MAC address where the IP address is configured.

When IP enabled devices want to communicate, the kernel of the originating device hands the IP packet to the network interface driver software and requests to deliver the IP packet to the recipient. The only way to communicate on a local Ethernet is by means of a MAC address, IP addresses are of no use there. To find out the MAC address of the recipient with the proper IP address, the network driver for the interface on the origination side will send an ARP request. An ARP request is a broadcast: it is sent to any computer on the local network. The computer that has the requested IP address will now answer back with its MAC address. The sender then has al the information needed to transmit the packet. Also, the MAC and IP addres are stored in a local cache for future reference.

The command **arp** can be used to show the ARP cache. Example:

```
# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.8.126               ether   00:19:bb:2e:df:73   C                     wlan0
```

The cache can be manipulated manually, for example if a host is brought down you might want to remove it's arp entry. Normally you do not need to bother as the cache is not overly persistent. See the man pages for more information on the **arp** command.

Note
Additionally, there exists the reverse ARP protocol (RARP). This protocol is used to allow an Ethernet (layer 2) device which IP address(es) it has configured. ARP: broadcast IP and receive MAC. RARP: broadcast MAC and receive IP.

Wireless networking

iw

iw is used to configure wireless devices. It only supports the nl80211 (netlink) standard. So if **iw** doesnt see your device, this might be the reason. You should use **iwconfig** (from the *wireless_tools* package) and **iwlist** to configure the wireless device. These are using the WEXT standard. *wireless_tools* is deprecated, but still widely supported.

Syntax:

```
# iw command
# iw [options] [object] command
```

Some common options:

dev This is an object and the name op the wireless device should follow after this option. with the command **iw dev** you can see the name of your device.

link This is a command and gets the link status of your wireless device.

scan This is a command and scans the network for available access points.

connect This is a command which lets you connect to an access point (essid), you can specify a channel behind it and/or your password.

set This is a command that lets you set a different interface/mode. For instance `ibss` if you want to set the operation mode to Ad-Hoc. Or set the power save state of the interface.

Examples:

```
# iw dev wlan0 link
# iw dev wlan0 scan
# iw dev wlan0 connect "Access Point"
# iw dev wlan0 connect "Access Point" 2432
# iw dev wlan0 connect "Access Point"  0:"Your Key"
# iw dev wlan0 set type ibss
# iw dev wlan0 ibss join "Access Point" 2432
# iw dev wlan0 ibss leave
# iw dev wlan0 set power_save on
```

iwconfig

iwconfig is similar to **ifconfig**, but is dedicated to the wireless interfaces. It is used to set the parameters of the network interface which are specific to the wireless operation (for example : the frequency). **iwconfig** may also be used to display those parameters, and the wireless statistics.

All parameters and statistics are device dependent. Each driver will provide only some of them depending on hardware support, and the range of values may change. Please refer to the man page of each device for details.

Syntax:

```
# iwconfig [interface]
# iwconfig interface [options]
```

Some common options:

essid Set the ESSID (or Network Name - in some products it may also be called Domain ID). With some cards, you may disable the ESSID checking (ESSID promiscuous) with off or any (and on to reenable it). If the ESSID of your network is one of the special keywords (off, on or any), you should use -- to escape it.

mode Set the operating mode of the device, which depends on the network topology. The mode can be Ad-Hoc (the network is composed of one cell only and is without an Access Point), Managed (the node connects to a network composed of multiple Access Points, with roaming), Master (the node is the synchronisation master or acts as an Access Point), Repeater (the node forwards packets between other wireless nodes), Secondary (the node acts as a backup master/repeater), Monitor (the node is not associated with any cell and passively monitors all packets on the frequency) or Auto.

Examples:

```
# iwconfig wlan0 essid any
# iwconfig wlan0 essid "Access Point"
# iwconfig wlan0 essid -- "any"
# iwconfig wlan0 mode Ad-Hoc
```

iwlist

iwlist is used to scan for available wireless networks and display additional information about them. The syntax is as follows:

```
# iwlist [interface] command
```

iwlist can display ESSID's, frequency/channel information, bit-rates, encryption type, power management information of other wireless nodes in range. Which information is displayed is hardware dependent.

Some useful options are:

scan[ning] Returns a list of ad-hoc networks and access points. Depending on the type of card, more information is shown, i.e. ESSID, signal strength, frequency. Scanning can only be done by root. When a non-root users issues the scan command, results from the last scan are returned, if available. This can also be achieved by adding the option **last**. Furthermore, the option **essid** can be used to scan for a specific ESSID. Depending on the driver, more options may be available.

keys/enc[ryption] List the encryption key sizes supported and list all the encryption keys set in the device.

Advanced Network Configuration and Troubleshooting (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.

Resources: Bird01, Drake00,

Key Knowledge Areas

- Utilities to manipulate routing tables
- Utilities to configure and manipulate ethernet network interfaces
- Utilities to analyse the status of the network devices
- Utilities to monitor and analyse the TCP/IP traffic

Terms and Utilities

- **/sbin/route**
- **/sbin/ifconfig**
- **/bin/netstat**
- **/bin/ping**
- **/bin/ping6**
- **/usr/sbin/traceroute**
- **/usr/sbin/traceroute6**
- **/usr/sbin/arp**
- **/usr/sbin/tcpdump**
- **/usr/sbin/lsof**
- **/usr/sbin/ss**

- **/usr/bin/nc**
- **/usr/bin/mtr**
- **/sbin/ip**
- **nmap**
- **wireshark**

Virtual Private Network

What Is A VPN

A VPN (Virtual Private Network) allows you to connect two or more remote networks securely over an insecure connection, for example over the public Internet. To do this an encrypted *secure tunnel* is created: all data will be encrypted before being sent over the insecure network. The resulting network connection acts and feels like a physical connection, but actually may traverse many physical networks and systems. Hence its name: "virtual".

VPNs are frequently used to save costs. In olden days physical connections had to be leased from telecom providers or you had to use POTS or ISDN lines. This was a costly business. Nowadays the Internet is omnipresent, and almost always available at a low fixed monthly price. However, the Internet can be sniffed and intruders might inspect and/or intercept your traffic. A VPN shields you from most of the problems you might have otherwise.

A use case might be to integrate LANs in several offices or branches. A user that works in the Los Angeles office hence can access the network services of the department in New York vice versa. In most cases, offices already have an Internet connection, so no additional investments need to be made.

VPN Types

There are many ways to implement a VPN, although most solutions either use IPSEC or SSL/TLS as their basis for encryption. Some companies use proprietary software implementations. Many routers have IPSEC based VPN support built in. A usable VPN can be built using a simple SSH tunnel or by using a more sophisticated dedicated solution. Some VPN implementations include:

- IPSEC
- VPND
- SSH
- Many Cisco Routers (or other proprietary implementations)
- SSL/TLS

This book will outline the implementations of OpenVPN and IPSEC. OpenVPN is covered separately in the chapter on System Security.

IPSEC

IPSEC provides encryption and authentication services at the IP (Internet Protocol) level of the network protocol stack. It replaces/enhances the regular level 3 IP layer so all packets are encrypted, including for example UDP packets. The IPSEC layer has been standardized by the IETF in RFCs 2401–2412. Implementing IPSEC is an option for IPv4 but is mandatory in IPv6 stacks.

In a regular IPv4 network you might set up dedicated IPSEC gateway machines to provide encrypted IP network connections when needed. IPSEC can run on routers, firewall machines, various application servers and on end-user desktop or laptop machines - any system that has an IP stack.

Using IPSEC is simple, as the protocol is built-in into the IP stack. But there are additional tasks in comparison with a regular IPv4 connection, for example encryption keys need to be exchanged between the end-points before an encrypted tunnel can be

set up. It was decided to handle this over a higher-level protocol, the Internet Key Exchange protocol (IKE). After IKE has done its work, the IP level services ESP and AH know which keys to use to do their work.

The full names of the three protocols that are used in an IPSEC implementation are:

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 protocols mentioned above. The IKE protocol ensures authentication of the peers and exchange of their symmetric keys. The IKE protocol is usually implemented by a user space daemon that uses port 500/udp.

Note
IPSEC standards define all three protocols, but in some contexts people use the term IPSEC to refer to just AH and ESP.

OpenSwan, formerly known as FreeS/WAN, is a complete IPSEC implementation for Linux 2.0 - 2.6 kernels. StrongSwan (also derived from FreeS/WAN) is another implementation that also supports the 3.x kernel. Both OpenSwan and FreeSwan implement all three protocols mentioned earlier. The Openswan implementation has several main parts:

- KLIPS (KerneL IPSec) which implements generic IPSEC packet handling, AH and ESP on the kernel level, for all kernels before version 2.5.47. KLIPS has been superseded by native IPSEC kernel support (NETKEY).
- NETKEY is the Kernel IPSec implementation included with the 2.6 kernel.
- Pluto (an IKE daemon) implements IKE, negotiating connections with other systems.
- various scripts provide an administrator interface to the machinery.

The config file contains 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:

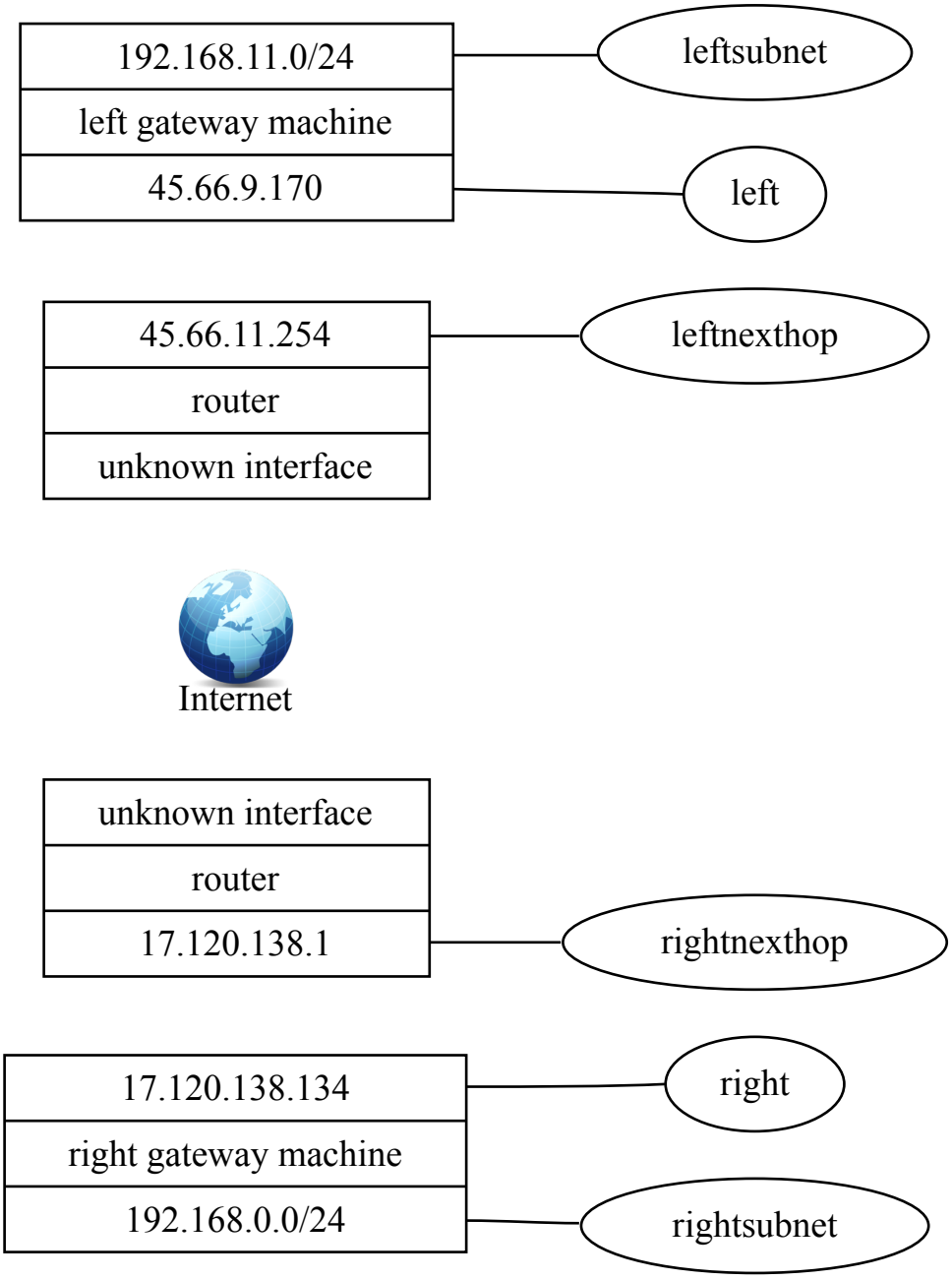
```
# basic configuration
config setup
        interfaces="ipsec0=eth0"
        klipsdebug=none
        plutodebug=none
        plutoload=%search
        plutostart=%search
        # Close down old connection when new one using same ID shows up.
        uniqueids=yes

# 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
```

In a typical setup you have two interconnected gateways that both run IPSEC and route packets. One of these gateways can be seen as 'the one on the left', the other as 'the one on the right'. Hence specifications are written in terms of `left` and `right` participants. There is no special meaning attached to either name, they are just labels - you might have defined the 'left' host to be the 'right' host and vice versa.

Normally, you would use the exact same configuration file on both sides. Interpretation of that file is done by checking the local configuration. For example if it was stated in the configuration file that the IP address for 'left' is 1.2.3.4, the software assumes that it runs on the left node if it finds that IP address configured on one of its network devices. The same file is interpreted differently on the other node, as that hosts configuration differs.

The `left`, `leftsubnet` and `leftnexthop` (and the `right...` counterparts) determine the layout of the connection:



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 keys can be generated on both client and server with the command:

```
# ipsec rsasigkey --verbose 2048 > rsa.key
```

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 routes traffic through the tunnel if need be.

`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.

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

Typing **ifconfig** without additional parameters displays the configuration for all network interfaces on the system. You might use this command to verify the configuration of an interface if the user experiences connectivity problems, particularly when their system has just been (re)configured.

When **ifconfig** is entered with an interface name and no other arguments, it displays the current values assigned to that particular 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 may be the case when the host can reach some hosts on its local subnet but is unable to reach other hosts, even if they are on the same subnet. **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; because the IP address is unfamiliar to the other hosts on the network, return packets will be directed to their default gateway (often leading to the internet) or even dropped. In this case, using **ifconfig** may 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 intermittent communication problems to your system. This type of configuration error cannot be discovered by **ifconfig**, because the error is on a remote host. IP conflicts like this can be discovered using the **arp** command, which will show two alternating MAC addresses for the same IP address.

The **ifconfig** command can be used to set up multihomed network device. There are two ways a host can be multihomed.

Two Or More Interfaces To The Same Network: Devices such as servers or high-powered workstations may be equipped with two physical interfaces to the same network for performance and/or reliability reasons. They will have two IP addresses on the same network with the same network ID.

Interfaces To Two Or More Different Networks: Devices may have multiple interfaces to different networks. The IP addresses will typically have different network IDs in them.

ping and ping6

The basic format of the ping or ping6 command on a Linux system is:

```
$ ping [-c count] host
$ ping6 [-c count] host
```

host The hostname or IP address of the remote host being tested. Note that you cannot ping from an IPv4 host to an IPv6 host or vice versa. Both ends need to use the same IP version.

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 high or packets are arriving out of order, there could be a network hardware or link 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 moderate 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 close to zero; there should be little or no packet loss and the packets should arrive in order. If these conditions are not met, there is a problem with the network hardware or with the links connecting them. 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 automatically added. This 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 and **traceroute6** are tools used to discover the gateways along a path. Path discovery is an essential step in diagnosing routing problems. Note that **traceroute6** is equivalent to **traceroute -6**

The **traceroute** command 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 lifetime 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 filtered by a firewall.

The **traceroute** command 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 MAC 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.

IP address conflicts are often the result of configuration errors including:

- assignment of the same static IP address by a network administrator
- assignment of a static IP address within a DHCP range (dynamic range) resulting in the same address being automatically assigned by the local DHCP server
- an error in the DHCP server
- a system coming back online after an extended period in stand-by or hibernate mode with an IP address that has been reassigned and is in use on the network.

Detection of duplicate IP addresses can be very hard even with **arpwatch**. IP address conflicts occur when two devices on a network are assigned the same IP address, resulting in one or both being disabled and losing connectivity until the conflict is resolved.

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.

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.

tcpdump

tcpdump is a program that enables network administrators to inspect every packet going through the network in real-time. This tool is typically used to monitor active connections or troubleshoot occasional network connectivity problems. In order to see traffic, however, the host running **tcpdump** must be somewhere along the path between two (or more) hosts exchanging traffic. This may prove to be difficult in a fully switched network. An easy solution is to run tcpdump on the host that needs to send or receive traffic. Another option is to configure a port on one of the switches where a copy of traffic from certain source and destination ports is sent; this is called a SPAN port.

tcpdump can generate a lot of output, so it is useful to narrow the scope of packets captured by specifying the interface you want to listen on using `-i`. In addition, you can specify the source, destination, protocol type and/or port number of the traffic you want to see joined by boolean AND and OR statements if necessary. An example command could be: `tcpdump -i eth0 src 10.10.0.1 and dst 10.10.0.254 and tcp port 80`). Other useful options are `-n` to turn of name resolution and `-w` to write captured packets to a file for later inspection (e.g. in Wireshark).

nmap

nmap is a versatile tool for network exploration and security auditing. Its main use is as a portscanner, which also can identify running services, versions of the running services and OS types.

An example of this command and output is:

```
$ nmap -A localhost

Starting Nmap 6.25 ( http://nmap.org ) at 2013-07-04 04:01 CDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00092s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 993 closed ports
```

```
PORT     STATE SERVICE   VERSION
22/tcp   open  ssh       OpenSSH 6.2p2 Debian 4 (protocol 2.0)
| ssh-hostkey: 1024 f6:bf:2d:57:41:1b:fe:fa:d2:10:e0:2d:c3:89:c1:80 (DSA)
| 2048 0d:11:fc:60:83:69:19:76:d1:fd:01:e5:7f:36:19:00 (RSA)
|_256 0a:7b:0c:3a:86:c1:f7:63:6e:fb:f0:3c:62:42:c1:58 (ECDSA)
25/tcp   open  smtp      Exim smtpd 4.80
| smtp-commands: linux.mailserver Hello localhost [127.0.0.1], SIZE 52428800, 8BITMIME,  ↩
    PIPELINING, HELP,
|_ Commands supported: AUTH HELO EHLO MAIL RCPT DATA NOOP QUIT RSET HELP
53/tcp   open  domain
| dns-nsid:
|_  bind.version: 9.8.4-rpz2+rl005.12-P1
80/tcp   open  http      Apache httpd 2.2.22 ((Debian))
|_http-title: Site doesn't have a title (text/html).
111/tcp  open  rpcbind   2-4 (RPC #100000)
| rpcinfo:
|   program version   port/proto  service
|   100000  2,3,4        111/tcp  rpcbind
|   100000  2,3,4        111/udp  rpcbind
|   100024  1          40817/udp  status
|_  100024  1          49956/tcp  status
389/tcp  open  ldap      OpenLDAP 2.2.X - 2.3.X
3000/tcp open  ntop-http Ntop web interface 4.99.3
Service Info: Host: linux.mailserver; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at http://nmap.org/ ↩
    submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.46 seconds
```

Note
Some of the **nmap** command options require root privileges, consult the `NMAP(1)` manpage for more information

wireshark

wireshark (known as Ethereal until a trademark dispute in the summer of 2006) is an open source network protocol analyzer. It allows you to examine data from a live network or from a capture file on disk. You can interactively browse the capture data, delving down into just that level of packet detail you need.

Wireshark has several powerful features, including a rich display filter language and the ability to view the reconstructed stream of a TCP session. It also supports hundreds of protocols and media types. A tcpdump-like console version named tethereal is also included. One word of caution is that Ethereal has suffered from dozens of remotely exploitable security holes, so stay up-to-date and be wary of running it with root privileges on untrusted or hostile networks (such as security conferences).

/usr/sbin/lsof

The **lsof** command lists all open files on a system. Since Linux treats everything as a file, it also shows open network sockets. It can be used to find open ports on a system as well as determining the origin of a network connection.

Options of the **lsof** used for network troubleshooting.

-i List IP sockets. The `-i` option also takes arguments to restrict the sockets listed, like `-i` *`tcp`* or `-i` *`192.168.1.2`*.

-n Do not resolve hostnames; can make **lsof** run faster.

-P Do not resolve port names; can make **lsof** run faster.

+P Resolve port name to protocol; default behaviour.

/usr/sbin/ss

The **ss** command can be used to investigate network sockets on a system, similar to netstat.

Useful options of the **ss** for network troubleshooting include:

-a List all sockets. This includes sockets in listening state.

-n Do not resolve port names.

-l Only display listening sockets.

-p Show processes using the sockets

-t Restrict output to TCP sockets

-u Restrict output to UDP sockets

/bin/netstat

Like **lsof**, **netstat** is a program to list open ports on a system. It looks for the standard ports, but it also finds custom ports opened by applications, for instance **netcat**.

Frequently used options are:

-a List all sockets.

-e Extended mode.

-inet List only IP connections

-l Only show listening sockets

-n Show IP numbers instead of resolving them to hostnames.

-p Show the PID number and name of the process that is holding the socket.

-r Show the kernel routing table. Is the same as the **route** command.

/usr/bin/nc

nc is used for establishing TCP and UDP connections between arbitrary ports on either end. After opening a port, it can listen for input, which can be passed through to another command for further processing. Note that you need to have administrative privileges on the system you're running this command on for opening a listening port below 1024. The command allows the user to set up, analyze and read connections between systems. It is a very useful tool in the troubleshooting toolbox. **nc** can be used to open up any port.

Because **nc** is capable of connecting and listening to any port it can be used to transfer files between systems that lack Samba, FTP or SSH etc.

if **nmap** is not available the **nc** command can be used to check for open ports: Run nc command with -z flag. You need to specify host name / ip along with the port range to limit and speedup operation. eg.

```
# nc -z localhost 1-1023
```

/usr/bin/mtr

The **mtr** is extremely helpful for troubleshooting network problems, because it combines the functionality of ping and traceroute. Rather than provide a simple outline of the route that traffic takes across the internet like traceroute, mtr collects additional information regarding the state, connection, and responsiveness of the intermediate hosts.

mtr can be used without any options. Just type: **mtr> host** to get a visual display of the path between you and the host and per-hop statistics. Like **ping**, **mtr** will continue to send ICMP packets indefinitely by default. Useful options are:

-n Do not reverse resolve IP addresses to hostnames

-c count Send count number of probe packets, then stop

/sbin/ip

The **ip** command is already discussed in the previous chapter.

Troubleshooting network issues (205.3)

Candidates should be able to identify and correct common network setup issues, to include knowledge of locations for basic configuration files and commands.

Key Knowledge Areas

- Location and content of access restriction files
- Utilities to configure and manipulate ethernet network interfaces
- Utilities to manage routing tables
- Utilities to list network states
- Utilities to gain information about the network configuration
- Methods of information about the recognised and used hardware devices
- System initialisation files and their contents (Systemd and SysV init process)
- Awareness of NetworkManager and its impact on network configuration

Terms and Utilities

- **ip**
- **ifconfig**
- **route**
- **ss**
- **netstat**
- `/etc/network/`, `/etc/sysconfig/network-scripts`
- **ping**, **ping6**
- **traceroute**, **traceroute6**
- **mtr**

- **hostname**
- System log files such as `/var/log/syslog` & `/var/log/messages`
- **dmesg**
- `/etc/resolv.conf`
- `/etc/hosts`
- `/etc/hostname`, `/etc/HOSTNAME`
- `/etc/hosts.allow`, `/etc/hosts.deny`

Introduction to network troubleshooting

It would be great if you would be able to open up a book and find an index there of all possible network problems and their solutions. But in practice that is impossible. There are way too many scenarios to describe and new technologies are surfacing constantly, creating new options - and new problems. However, almost every problem can be solved by applying knowledge and logic. In the case of network troubleshooting, this means mostly that you should determine how traffic *should* flow from source to destination and back, and checking step by step if you can see the traffic passing by using the tools described earlier. Also, make sure to check traffic in both directions if possible, as many network problems stem from the fact that traffic from source to destination may not be following the same path as traffic the other way around.

Key files, terms and utilities have already been described in Chapter 5 and other chapters. In this chapter we focus on the problem solving process and introduce a number of additional techniques, utilities and key files.

An example situation

You are browsing the Internet on a PC. It is connected to the Internet via a local area network and a firewall. Suddenly, you can not access your favourite webpage anymore. It could be the network, the firewall, the ISP, or even the browser.. what is a reasonable approach to find the problem and solve it?

Note
The focus of this section is on troubleshooting networks. Hence we will focus on network components. In a real situation there are many more components involved, for example the browser, operating system, local firewall setups etc. on which we will touch only briefly.

The first step is to assemble a list of all network components involved. The lenght of the list varies, depending on the complexity of the configuration and your personal knowledge. A simple list would at least contain this:

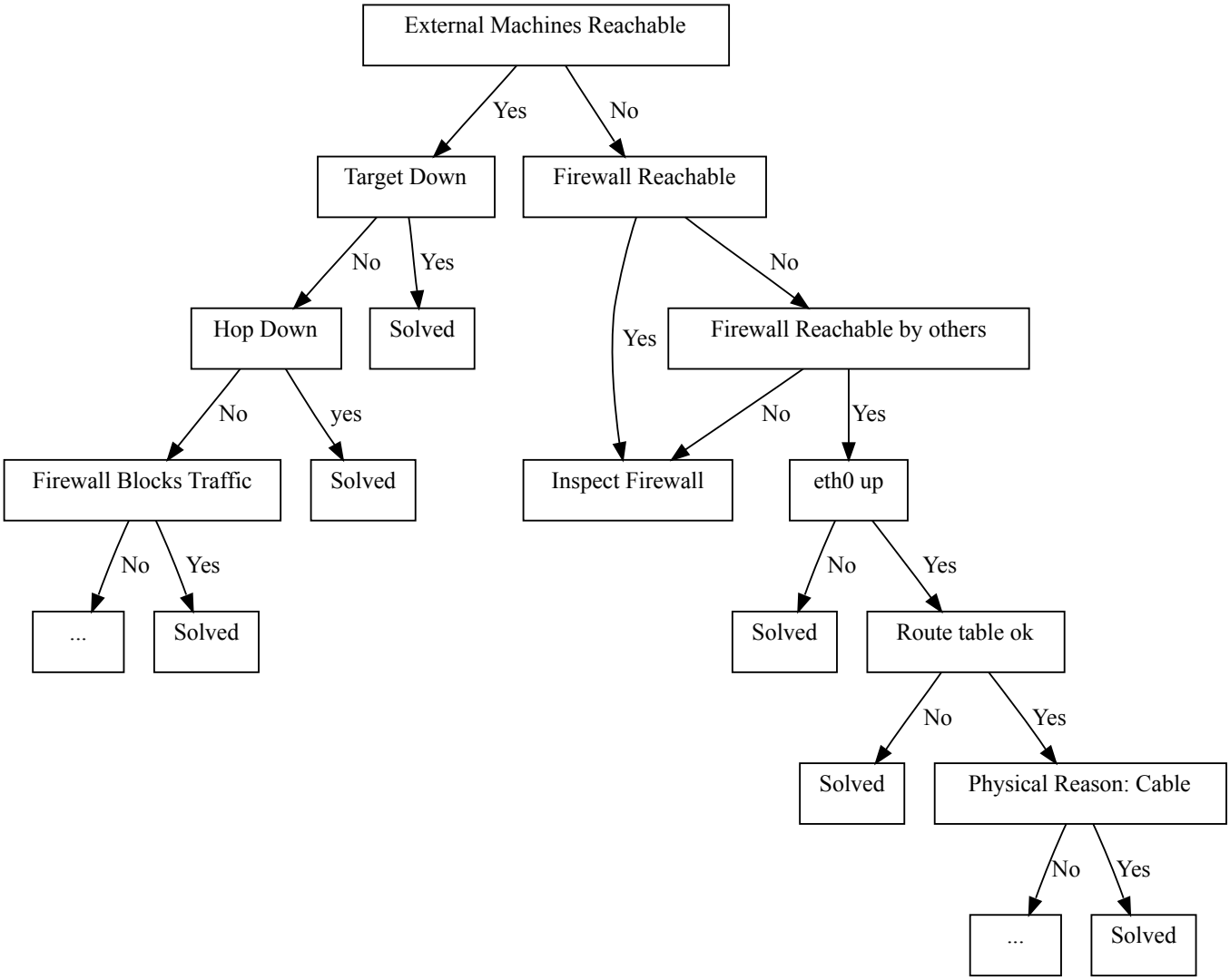
- The PC itself. It has a network interface that is connected to the LAN (eth0 in most situations);
- The firewall. It has two interfaces: its eth0 interface which is connected to the LAN and the eth1 interface which is connected to the router that in turn is connected to the an ISP who provides Internet connectivity;
- The site you are trying to reach, connected to the Internet.

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 through 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 gives an example of step-by-step troubleshooting.



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 to jump to conclusions if your **ping** reports no connectivity - your firewall could be blocking ICMP echo-requests and replies.

2 Is the machine we are trying to reach, the target, down? This is a feasible theory if for example other sites can be reached. 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 router on the route (a "hop") down ? Use **traceroute** to find out what the hops are between you and the target host. The route from a 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 blocked traffic to and/or from that site.

7 Inspect the firewall. If the problem seems to be on the firewall, test the interfaces on the firewall, inspect the firewalling rules, check the cabling etc.

8 Is our eth0 interface up? This can be tested by issuing the command **ifconfig eth0**.

9 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**.

10 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.

Name resolution problems

There are even more options why the connection fails:

Name resolution is the translation of a hostname into an IP address. If a user tries to connect to a machine based on the hostname of that machine and the hostname resolution doesn't function properly then no connection will be established at all.

The file `/etc/resolv.conf` contains the IP addresses of the nameservers. The nameservers are the servers that do the name resolution for a external network. For small (local) networks a local lookup table can be made by using the `/etc/hosts` file. This file contains a list of aliases or FQDN (fully qualified domain name) (or both) per IP address.

You can check name resolution with the commands **/usr/bin/dig** (dig is an acronym for Domain Information Groper) or **/usr/bin/host**. Both of these commands return the IP address associated with the hostname.

```
$ host ns12.zoneedit.com
ns12.zoneedit.com has address 209.62.64.46
```

```
$ dig zonetransfer.me

; <<> DiG 9.8.3-P1 <<>> zonetransfer.me
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6133
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1

;; QUESTION SECTION:
;zonetransfer.me.   IN  A

;; ANSWER SECTION:
```

```
zonetransfer.me.  7142  IN  A 217.147.180.162

;; AUTHORITY SECTION:
zonetransfer.me.  7142  IN  NS  ns12.zoneedit.com.
zonetransfer.me.  7142  IN  NS  ns16.zoneedit.com.

;; ADDITIONAL SECTION:
ns12.zoneedit.com.  7116  IN  A 209.62.64.46

;; Query time: 1 msec
;; SERVER: 213.154.248.156#53(213.154.248.156)
;; WHEN: Thu Jul  4 10:59:55 2013
;; MSG SIZE  rcvd: 115
```

dig is the swiss army knife of name resolving and has a lot of options. It provides elaborate output. The **host** command offers a fast and convenient way to seek out an IP address for a host known by its name.

The hostname of a machine itself is stored in a file called `/etc/hostname` or `/etc/HOSTNAME` for Debian based systems. On Fedora systems the name is stored in the file `/etc/sysconfig/network`. For all systems the hostname can be found with the command **/bin/hostname**. When given no argument, this command gives replies with the hostname of the machine. In case an argument is given along with the command, the hostname of the machine will be changed.

Incorrect initialization of the system

$ Another possible cause of network problems can be the incorrect initialization of the system. To find any initialization errors check out the file `/var/log/messages` or read the kernel ring buffer by using the **/bin/dmesg** command.

Security settings

Security settings can also be a source of connection problems. The server may have blocked access from or allow access from certain clients using the `/etc/host.deny` resp. `/etc/host.allow`

Network configuration

For instance if fixed network settings were copied over from another site and not adapted to the local situation. You can check these settings in the files in the directory `/etc/sysconfig/network-scripts` for Fedora based systems or in the file `/etc/network` for Debian based systems.

NetworkManager

NetworkManager is a GUI based tool to manage your networkconnections. NetworkManager is a also a service that is able to report network changes. The purpose of NetworkManager is to simplify the use of configuring your network within Linux.

Typically the user settings will be stored in: **/home/$user/.gconf/system/networking/connections** The system settings are stored in: **/etc/Networkmanager/ /etc/NetworkManager/system-connections** Be aware of the fact that NetworkManager will overwrite any configuration changes made to networksettings.

There is also an option to configure your NetworManager on the commandline. It is called nmcli and you can find it at **/usr/bin/nmcli**.

Questions and answers

Networking Configuration

1. *Is configuring the loopback interface necessary for a system without networking?*

   Yes, as the loopback interface is used by various local services during normal operation. Use of loopback interface [122]

2. *What is the purpose of configuring a default gateway?*

   In order to route networking traffic to destinations other than the local network, the default gateway is used. Defining a default gateway [123]

3. *Which command is used to scan for available wireless networks?*

   iwlist is the command by which to scan for available wireless networks. Scan for wireless networks

4. *What do VPN implementations - like IPSEC, VPND, SSH and (many) Cisco Routers - have in common?*

   They all create a *secure tunnel* through a possibly insecure network (like the internet). Common in VPN's [128]

5. *Name the three protocols which are used in an IPSEC implementation.*

   ESP (Encapsulating Security Payload), *AH* (Authentication Header) and *IKE* (Internet Key Exchange). Protocols used by IPSEC [128]

6. *Which tool is the first to use in order to detect mulitple use of your IP address in the same network?*

   Using **arp** one can determine which MAC-addresses are using your IP-address. Detecting IP addresses [125]

7. *What ICMP message will be sent back to a **traceroute** client by every system along the path to the destination specified?*

   An ICMP *TIME_EXCEEDED* reply is sent back to the packet's source address. This is because **traceroute** sents out a stream of packets to the destination, starting with the **TTL**-field set to 1 and incrementing it by one every time until the destination replies. Traceroute and ICMP [134]

Chapter 6

System Maintenance (206)

This topic has a total weight of 7 points and contains the following objectives:

Objective 206.1; Make and install programs from source (4 points) Candidates should be able to build and install an executable program from source. This objective includes being able to unpack a file of sources.

Objective 206.2; Backup Operations (3 points) Candidates should be able to use system tools to back up important system data.

Objective 206.3; Notifying users on system-related issues (1 point) Candidates should be able to notify the users about current issues related to the system.

Make and install programs from source (206.1)

Candidates should be able to build and install an executable program from source. This objective includes being able to unpack a file of sources.

Resources: the man pages of **tar(1)**, **gzip(1)**, **bzip2(1)**, **xz(1)**, **make(1)**, **uname(1)** and **install(1)**.

Key Knowledge Areas

Unpack source code using common compression and archive utilities

Understand basics of invoking make to compile programs

Apply parameters to a configure script

Know where sources are stored by default

Terms and Utilities

- `/usr/src`
- **gunzip**
- **gzip**
- **bzip2**
- **xz**
- **tar**

- **configure**
- **make**
- **uname**
- **install**
- **patch**

Unpacking source code

Most Open Source software is distributed as compressed tarballs containing source code and build scripts to compile and install the software. Preferably the source is extracted and compiled in `/usr/src/` but any location will do.

These tarballs are generally compressed using gzip, bzip2 or xz. GNU tar supports these compression formats, and makes it easy to decompress such files. For example, to unpack a gzip compressed tarball:

```
$ tar zxvf /path/to/tarball.tar.gz
```

The `z` option tells tar to use the gzip algorithm, and the `x` option tells tar to extract the file. To extract a bzip2 compressed file use GNU tar's `j` option:

```
$ tar jxvf /path/to/tarball.tar.bz2
```

To extract a xz compressed file use GNU tar's `J` option:

```
$ tar Jxvf /path/to/tarball.tar.xz
```

Although GNU tar supports these compression alghorithms, several other tar implementations don't. To extract compressed tarballs on machines with such tar implementations, you first need to decompress the file, and then extract the contents.

For a gzip compressed tarball:

```
$ gunzip /path/to/tarball.tar.gz
```

For a bzip2 compressed tarball:

```
$ bunzip2 /path/to/tarball.tar.bz2
```

For a xz compressed tarball:

```
$ unxz /path/to/tarball.tar.xz
```

As an alternative, you can also use the '-d' (decompress) argument to the **gzip**, **bzip2** and **xz** commands.

After decompression, you can extract the contents by calling tar without giving a compression argument:

```
$ tar xvf /path/to/tarball.tar
```

Building from source

Usually the build scripts are generated using GNU autoconf and automake. automake is used to generate GNU Coding standard compliant Makefile.in files. autoconf produces self-contained configure scripts which can then be used to adapt, build and install the software on the target system.

The usual way of building and installing software from source looks something like this:

```
$ tar zxvf snow-2.0.3.tar.gz
$ cd snow-2.0.3
$ ./configure
$ make
$ su
# make install
```

The **./configure** command will check for both optional and mandatory dependencies. It will also adapt the source code to the target system (think system architecture, installed libraries, compiler flags, install directories, ...). If an optional dependency is missing, it will disable compilation to that dependency. In the case of missing required dependencies, it will print the error and exit.

According to GNU standards, the commands above would install the "snow" application under `/usr/local`. If you want to install the application under some other directory structure, for example `/opt`, the configure command would look like:

```
$ ./configure --prefix=/opt
```

Try **./configure `--help`** to see all possible configure arguments.

The configure command also generates Makefiles which **make** uses to compile and install the software. The `Makefile` usually contains several "build targets" which you can call by giving them as an argument to make. Often used targets include "all" which is usually the default action, "clean" to clean up (remove) built object files from the source tree, and "install" to install the software after the build stage.

It is possible to install the software in a location other than the build directory. This is for useful if you want to build the software on one machine, and install it on another:

```
$ tar zxvf snow-2.0.3.tar.gz
$ cd snow-2.0.3
$ ./configure --prefix=/opt/snow
$ make DESTDIR=/tmp/snow_tmp install
```

This technique is often used to build software on one machine, and install it onto multiple others. In the above case, the `/tmp/snow_tmp` directory would only contain files installed by snow-2.0.3.

If not cross-compiling for another platform, configure will use **uname** to guess what machine it 's running on. Configure can usually guess the canonical name for the type of system it's running on. To do so it runs a script called `config.guess`, which infers the name using the **uname** command or symbols predefined by the C preprocessor.

```
$ cd /tmp/snow_tmp
$ tar zcf ../snow_2.0.3_built.tgz opt/
```

The c option (as opposed to the previously mentioned x option) to tar tells it to create a new archive.

If you copy the resulting tarball to the target machines in, let's say, the `/tmp` directory, you can execute the following commands to install the snow software into `/opt/snow`:

```
# cd /
# tar zxvf /tmp/snow_2.0.3_built.tgz
```

As alternative to using the make script you can use the install utility to copy binaries to the required location. When you copy the files with **install**, permissions and owner/group information will be set correctly. If the destination file(s) already exists they will be overwritten. But you can have **install** create a backup of these files by using the `-b` argument or using the `VERSION_CONTROL` enviroment variable.

Patch

patch is a program that updates files. In short: apply a diff file on a original file. **patch** takes a patchfile containing a difference listing produced by the diff program and applies those differences to one or more original files, producing patched versions.

Normally the patched versions are put in place of the originals. The names of the files to be patched are usually taken from the patchfile, but if there's just one file to be patched it can specified on the command line as originalfile.

Here is an example on how **patch** works:

To apply a patch

```
$ patch -p1 < /path/to/path-x.y.z
```

To revert a patch

```
$ patch -R -p1 < /path/to/patch-x.y.z
```

Note
Patching is not part of the LPIC-2 objectives, but is included here because it is used frequently when you build from source.

Backup Operations (206.2)

Candidates should be able to use system tools to back up important system data.

References: LinuxRef07, Wirzenius98.

Key Knowledge Areas

Knowledge about directories that have to be include in backups

Awareness of network backup solutions such as Amanda, Bacula, Bareos and BackupPC

Knowledge of the benefits and drawbacks of tapes, CDR, disk or other backup media

Perform partial and manual backups

Verify the integrity of backup files

Partially or fully restore backups

Awareness of Bareos

Terms and Utilities

- **/bin/sh**
- **dd**
- **tar**
- `/dev/st*` and `/dev/nst*`
- **mt**
- **rsync**

Why?

Making backups is the sysadmins Pavlov reaction to having at least one system to administer. But do we still need backups? After all, nowadays data is often stored on RAID cabinets or "in the cloud".

Well, it depends. First of all superredundant storage will only protect you against one reason for data-loss, namely hardware failure. But not against human error, software bugs and natural disasters.

After all, 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. Nature may not be evil, but, nevertheless, can be very destructive sometimes. The most reliable thing is hardware, but it still breaks seemingly spontaneously, often at the worst possible time.

To protect yourself against any of these threats, a very cheap and simple control is available: to make regular backups. There is a very low break-even point between the costs of data loss and those of making backups. In fact, the control used to be so cheap that employing it became a solid best practice.

But please note that there are situations in which making a backup is not necessary. An example is a math-cluster. The results of its calculation are important and you probably will make backups of these. But cluster nodes themselves are discardable. Might a node fail you can simply replace it with a freshly installed one.

So, in conclusion: making backups should always be the result of proper risk-analysis. But in the overwhelming majority of cases they are a cheap way of ensuring your data availability and in some cases your data integrity.

What?

What you need to backup should be the result of proper risk analysis. In olden days most system administrators simply backed up as much as possible - that may not be such a good idea anymore as it takes a lot of time to backup and restore large quantities of data. In all situations you do not need to backup certain parts of the Linux filesystem anyway, for example the `/proc` and `/sys` filesystems. They only contain data that the kernel generates automatically, 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 - do a proper risk analysis.* 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 or they may be created when you reinstall a system.

The obvious things to back up are user files (`/home`) and system configuration files (`/etc`, but possibly other things scattered all over the filesystem).

Generally it is good idea to backup everything needed to rebuild the system as fast as required after a failure - and nothing else. It is often quite difficult to find out what to backup and what not, hence the sysadmins paradigm: whilst in doubt, back it up.

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.

To determine your minimal backup frequency consider the amount of data you are confident you can afford to loose without severe consequences for you and/or your company. Then consider the minimal timespan needed to create that amount of data. You should make at least one backup during that timespan. Practical considerations like the time it takes to make a backup and whether or not you can do any work when a backup is being made, will restrict the maximum frequency. In the past systems were mainly used during the day and not at all in weekends. Hence many companies made incremental backups during the night and a full backup during the weekend. However, nowadays many systems need to be used 24 x 7 which creates the need for alternate strategies, for example creating snapshots on disk, which can subsequently be backed up on tape while the production systems continue their work on the live filesystem.

Backups can take several hours to complete, but, for a successful backup strategy, human interaction should be minimal. Backups are mostly made to disk or on tape. In the latter case, the human interaction can be further reduced by having tape robots that

change the tapes for you. However, make sure to store tapes off-site so any disaster will not take out your backups. And be sure to protect sensitive data on tapes you store off-site, for example by encrypting the data stored on them.

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 you made that backup on your only medium you lost your data. 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.

You will need a written plan, the backup plan, which describes what is backed up, how to restore what is being backed up an any other procedures surrounding the backup. 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.

One important thing to be aware of, as noted in the previous paragraphs, you need to be able to rebuild your system (or restore certain files) as fast as required. In some enviroments restore times can be hours because of slow network lines or other causes. The time lost may be too much and can defeat the purpose of the backup system. Other solutions for continuity of service like cluster/failover systems are recommended.

There are different types of backup media, each with their own benefits and drawbacks. The medium of choice however will often be made based on total cost. The main types are:

Tape
Disk
Optical Media
Remote/Network

Tape

Tape is one of the most used mediums for backup in enterprise environments. It is low cost and because tapes store passively they have a low chance for failure and consume little power on standby. A disadvantage of tape is that it is a streaming medium which means high access times, especially when a tape robot is used for accessing multiple tapes. Bandwidth can be high if data is provides/requisted in a continuous stream. Tape is especially suitable for long term backup and archiving. If a lot of small files have to be written or restored the tape will need to be stopped and started and may even need to be partially rewound frequently to allow processing by the restoring operating system. This may consume excessive time. In those cases tape is not the best medium.

Disk

Local disk storage is hardly used for backup, (though it is used for network storage, see below). The advantages are high bandwidth, low latency and a reasonable price compared to capacity. But it is not suitable for off-site backup (or the disk has to be manually disconnected en transported to a safe location). And since disks are always connected and running, chances for failure are high. Though not suitable for off-site backup it is sometimes used as intermediate (buffer) medium between the backup system and an off-site backup server. The advantage is that fast recovery of recent files is possible and the production systems won't be occupied by long backup transfers. Another option, suitable for smaller systems, is using cheap USB bus based portable disks. These can be spun down and powered off when not in use, while still retaining their data. They can contain many terabytes of data and can be taken off-site easily. Also, modern disks employ fast USB protocols that reduce backup- and restore-time.

Optical Media

Optical media like CDROM and DVDR disk are mostly used to backup systems which don't change a lot. Often a complete image of the system is saved to disk for fast recovery. Optical disks are low cost and have a high reliability when stored correctly. They can easily be transported off-site. Disadvantages are that most are write-once and the storage capacity is low. Access time and bandwidth are moderate, although mostly they have to be handled manually by the operator.

Remote/Network storage

Network storage is mostly remote disk storage (NAS or SAN). Using data protection techniques like RAID, the unreliability of disks can be reduced. Most modern network storage systems use compression and deduplication to increase potential capacity. Also most systems can emulate tape drives which makes it easy to migrate from tape. The cost of the systems can be high depending on the features, reliability and capacity. Also power costs should be considered because a network storage system is always on. This type of medium is thus not preferred for long time backup and archives. Access time and bandwidth can differ and depend on infrastructure, but are mostly high.

Backup utilities

Rsync

Rsync is a utility to copy/synchronise files from one location to the other while keeping the required bandwidth low. It wil look at the files to copy and the files already present at the destination and uses timestamps, filesize and an advanced algorithm to calculate which (portions of) files need to be transferred. Source and destination can be local or remote and in case of a remote server SSH or **rsync** protocol can be used for network transfer. **Rsync** is invoked much like the **cp** command. Recursive mode is enabled with `-r` and archive with `-a`. A simple example to copy files from a local directory to a remote directory via SSH:

```
rsync -av -e ssh /snow remote:/snow
```

Note
By default **rsync** copies over (relevant parts of) changed files and new files from a remote system. It does *not* delete files that were deleted on the remote system. Specify the option `--delete` if you want that behaviour. Also note that permissions will be copied over correctly, but if your local UIDs/GIDs do not match te remote set you may end up with incorrect local permissions still.

---

tar

The **tar** utility is used to combine multiple files and directories into a continous stream of bytes (and revert the stream into files/directories). This stream can be compressed, transferred over network connections, saved to a file or streamed onto a tape device. When reading files from the stream, permissions, modes, times and other information can be restored. **Tar** is the most basic way for transferring files and directories to and from tape, either with or without compression.

An example of extracting a gzipped **tar** archive, with verbose output and input data read from a file:

```
tar xvzf snow.tgz
```

Extracting a **tar** archive from a scsi tape drive:

```
tar xvf /dev/st0
```

Creating a archive to file from the directory `/snow`:

```
cd /; tar cvf /tmp/snow.tar snow
```

By default the **tar** utility uses (scsi) tape as medium. As can be seen in the example above scsi tape devices can be found in `/dev/st*` or `/dev/nst*`. The latter one is a non rewinding tape, this means that the tape does not rewind automatically after each operation. This is an important feature for backups, because otherwise when using multiple **tar** commands for backups any backup but the last would be overwritten by the next backup.

Tapes can be controlled by the **mt** command (magnetic tape). The syntax of this command is: **mt [-h] [-f device] command [count]**. The option -h (help) lists all possible commands. If the device is not specified by the -f option, the command will use the environment variable TAPE as default. More information can be found in the manual pages.

dd

Using the **dd** utility, whole disks/partitions can be transferred from/to files or other disks/partitions. With **dd** whole filesystems can be backed-up at once. **dd** will copy data at byte level. Common options to **dd** are:

`if`, input file: for the input file/disk/partition
`of`, output file:
`bs`, block size: size of blocks used for transfer, can be optimised depending on used hardware
`count`, number of blocks to transfer (`dd` will read until end-of-file otherwise)

An example of **dd** usage to transfer a 1GB partition to file:

```
dd if=/dev/hda1 of=/tmp/disk.img bs=1024 count=1048576
```

cpio

The **cpio** utility is used to copy files to and from archives. It can read/write various archive formats including **tar** and zip. Although it predates `tar` it is less well known. **cpio** has three modes, input mode (*`-i`*) to read an archive and extract the files, output mode (*`-o`*) to read a list of files and compress them into an archive and pass-through mode (*`-p`*) which reads a list of files and copies these to the destination directory. The file list is read from *`stdin`* and is often provided by `find`. An example of compressing a directory into a cpio archive:

```
%cd /snow; find . | cpio -o > snow.cpio
```

Backup solutions

Complete backup solutions exist which help simplify the administration and configuration of backups in larger environments. These solutions can automate backup(s) of multiple servers and/or clients to multiple backup media. Many different solutions exist, each with their own strengths and weaknesses. Below you'll find some examples of these solutions.

AMANDA, the Advanced Maryland Automatic Network Disk Archiver, is a backup solution that allows the IT administrator to set up a single master backup server to back up multiple hosts over network to tape drives/changers or disks or optical media. Amanda uses native utilities and formats (e.g. **dump** and/or GNU **tar**) and can back up a large number of servers and workstations running multiple versions of Linux or Unix.

Bacula is a set of Open Source, enterprise ready, computer programs that permit you (or the system administrator) to manage backup, recovery, and verification of computer data across a network of computers of different kinds. Bacula is relatively easy to use and efficient, while offering many advanced storage management features that make it easy to find and recover lost or

damaged files. In technical terms, it is an Open Source, enterprise ready, network based backup program. According to Source Forge statistics (rank and downloads), Bacula is by far the most popular Enterprise grade Open Source program.

Bareos is a fork of the project Bacula version 5.2 and was started because of rejection and neglect of community contributions to the Bacula project. The Bareos backup program is open source and is almost the same as Bacula, but is does have some additional features like LTO hardware encryption, bandwidth limitation and new practical console commands. One focus in Bareos's development is keeping the obstacles for newcomers as low as possible. Because newcomers are usually overwhelmed by configuration options, the Bareos project offers package repositories for popular Linux distributions and Windows.

BackupPC is a high-performance, enterprise-grade system for backing up Linux, WinXX and MacOSX PCs and laptops to a server's disk. BackupPC is highly configurable and easy to install and maintain.

Notifying users on system-related issues (206.3)

Candidates should be able to notify the users about current issues related to the system.

Key Knowledge Areas

Automate communication with users through logon messages

Inform active users of system maintenance

Terms and Utilities

- `/etc/issue`
- `/etc/issue.net`
- `/etc/motd`
- **wall**
- **/sbin/shutdown**
- **/bin/systemctl**

Resources: How to use systemctl;

The `/etc/issue`, `/etc/issue.net`, and `/etc/motd` files

The `/etc/issue`, `/etc/issue.net` and `/etc/motd` files are used to send simple messages to users that log in to the system. The `/etc/motd` is used to display a message after the user has authenticated successfully. The `/etc/issue` on the other hand is used to send a message to the user before they login. This message is only displayed to users that log in using the console and will often contain some information about the system like kernel version and architecture. The `/etc/issue.net` file has the same purpose as the `/etc/issue` file but is used for insecure logins using **telnet**. It is possible to use the `/etc/issue.net` file for **ssh** as well. For this to work you need to add or modify the following line in `/etc/ssh/sshd_config`:

```
Banner /etc/issue.net
```

When using `/etc/issue.net` with ssh you should note that the special sequences may not work.

The wall command

wall is used to broadcast a message of at most 22 lines to all interactive terminals. By default the command can be used by any user, but often is reconfigured so only root can use it. A user not wishing to receive broadcast messages may use the **mesg** to write disable their terminals. The broadcaster may use the **finger** command to see which terminals are write disabled.

You can specify two options with the **wall** command: the `-n` option, which only works for the root user and the message itself. The `-n` suppresses the standard broadcast banner and replaces it with a remote broadcast banner. This option has meaning only when **wall** is used over the **rpc.walld** daemon. The second argument, the message itself can also be typed on `stdin`, in which case it must be terminated with an EOF (end of file, in most cases Ctrl-D).

The shutdown command communication.

As its name suggests, the **shutdown** command is used to shutdown a server gracefully, stepping down through the run level kill scripts, and optionally halting, or rebooting the server. The shutdown command itself is not discussed here, and this small section explains only the communicative steps that **shutdown** takes before, and during the system shutdown.

The last argument to the **shutdown** may optionally be used to broadcast some custom message explaining the purpose of the shutdown, and when it is expected to be returned to production. For example:

```
# shutdown -H +10 Server halting in 10 minutes for change change number. Expected up at ↩
    00:00:00.
```

Shutdown can be used with the `-k`. This makes **shutdown** do a 'dry-run': it emulates the shutdown, but does NOT shut down the system.

When you use the **-k** options you can append broadcast messages as part of the command line too, like in a "real" **shutdown**.

> **Note**
> Please note that **shutdown** `-k` will still temporarily disallow user logins as it will create the `/etc/nologin` file. It will be removed after the 'dry run' but your users will not be able to log in into the system as long as it is there.

In the event that a running **shutdown** needs to be cancelled, the **shutdown** may be called with the `-c` option, and again a broadcast message added to the command line to inform the users of the U-turn. As with all forms of message broadcasts, the receiving terminals must be write enabled.

The systemctl command communication,

The **systemctl** the central management tool for the systemd init system. **systemctl** can be used to manage services, system states (runlevels) and config files.

Managing services

Starting and stopping services

Where you used the **service** command in sysVinit you will now use the systemctl command to manage services. If you are using a non-root user to run the command you will have to use sudo. The following example shows starting a service using the **start** command:

```
$ systemctl start application.service
```

Because systemd knows you are running the system management commands on services you can also leave the .service suffix. For clarity we will keep using the suffix in the commands.

```
$ systemctl start application
```

For some programs it is possible to start the application multiple times with different configuration files. In this case you can pass the name of the config file to the command using the @-sign. For example, to start the openvpn service twice with different configuration files you can use the following commands:

```
$ systemctl start openvpn@config1.service
$ systemctl start openvpn@config2.service
```

Because they changed the order of the parameters for the systemctl command you can also start and stop multiple services at once: The following example show stopping multiple services using the **stop** command:

```
$ systemctl stop application1.service application2.service
```

Restarting and reloading services

For restarting a service you can use the **restart** command:

```
$ systemctl restart application.service
```

If an application is able to reload it's configuration you can also use the **reload** command:

```
$ systemctl reload application.server
```

If you not sure if a service can reload it's configuration you can also use the **reload-or-restart** command. This will reload is configuration if it is available, else it will restart the application:

```
$systemctl reload-or-restart application.service
```

Enabling and disabling service

The previous commands are useful for starting and stopping services in the current session. If you want a service to start at boot, for which sysVinit use the **chkconfig** command, you have to enable them using **systemctl**:

```
$ systemctl enable application.service
```

This **enable** command will create a symbolic link from this systems copy of the service file (which is usually found in /lib/systemd/system or /etc/systemd/system) to the directory where systemd looks for autostart files (usually /etc/systemd/system/-some_target.target.wants). To disable a service from start at boot you use the **disable** command:

```
$ systemctl disable application.service
```

This will remove the symbolic link that indicate that the service should start at boot.

Note
Remember that enabling a service will not start it in the current session. To start and enable a service you will need to issue both the start and enable command

---

Checking the status of services

To check the current status of a service you can use the **status** command:

```
$ systemctl status application.service
```

This command will give you the current state of the service, the cgroup hierarchy, and the first few log lines. It gives you a nice overview of the current status, and notifying you of any problems. For example the following output for the sshd service:

```
$ systemctl status sshd.service
  ● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset:  ↩
         enabled)
     Active: active (running) since Fri 2016-12-16 08:18:17 CET; 4h 58min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 1033 ExecStart=/usr/sbin/sshd $OPTIONS (code=exited, status=0/SUCCESS)
   Main PID: 1067 (sshd)
      Tasks: 1 (limit: 4915)
     CGroup: /system.slice/sshd.service
             `-1067 /usr/sbin/sshd

  Dec 16 08:18:17 localhost.localdomain systemd[1]: Starting OpenSSH server daemon...
  Dec 16 08:18:17 localhost.localdomain systemd[1]: sshd.service: PID file /var/run/ ↩
      sshd.pid not readable (yet?) after start: No such file or directory
  Dec 16 08:18:17 localhost.localdomain sshd[1067]: Server listening on 0.0.0.0 port  ↩
      22.
  Dec 16 08:18:17 localhost.localdomain sshd[1067]: Server listening on :: port 22.
  Dec 16 08:18:17 localhost.localdomain systemd[1]: Started OpenSSH server daemon.
```

There are also commands available for checking specific states which can be particularly useful for using them in scripts. For example to check if as service is currently active you can use the **is-active** command:

```
$ systemctl is-active application.service
```

This command will show if the application is active or inactive. If the application is active it will return an exit code of “0”. To see if an application is enabled you can use the **is-enabled** command:

```
$ systemctl is-enabled application.service
```

This command will return wheter the application is enbled or disable and will return an exit code of “0” if the application is enabled. To check if an application has failed you can use the **is-failed** command:

```
$ systemctl is-failed application.service
```

This command wil return active if the application is running, failed if an error has occured, and inactive or unknown the the service was intentionally stopped. It will return an exit code of “0” if the service has failed.

System state overview

The commands so far have been useful for managing single services. For exploring the the current state of the system there ar a number of systemctl commands that provide more information.

Listing current units

To get a list of all the active units that systemd knows about you can use the **list-units** command:

```
$ systemctl list-units
```

This command only shows a list of the currently active units. Then output has the following columns:

```
UNIT: The systemd unit name
LOAD: Whether the unit's configuration has been parsed by systemd. The
      configuration of loaded units is kept in memory.
ACTIVE: A summary state about whether the unit is active. This is usually a
        fairly basic way to tell if the unit has started successfully or not.
SUB: This is a lower-level state that indicates more detailed information
     about the unit This often varies by unit type, state, and the actual
     method in which the unit runs.
DESCRIPTION: A short textual description of what the unit is/does/
```

Because the **list-units** command only shows the active units by default, all of the entries above will show "loaded" in the LOAD colymn and "active" in the ACTIVE column. This is also the default behaviour of systemctl when called without additional commands, so you will see the same output if you call systemctl without arguments.

```
$ systemctl
```

You can also tell systemctl to output different information by adding additional flags. For instance, to show all units that systemd has loaded, whether they are active or not, you can use the `--all` flag:

```
$ systemctl list-units --all
```

This will show all units that systemd loaded or attempted to load, regardless of it's current state on the system. It is also possible to filter units by the current state, for this you can use the `--state=` flag. You will have to keep the `--all` flag so that systemctl allows non-active units to be disaplayed. For example, if you wish to see all inactive units you can issue the following command:

```
$ systemctl --list-units --all --state=inactive
```

Another filter you can use is the `--type=` flag. You can tell systemctl to only show the unit types you are interested in. For example, to only show active service units you can use the following command:

```
$ systemctl --list-units --type=service
```

Listing unit files

The **list-units** command we just used only shows units that systemd has attempted to load into memory. Systemd will only read units that it thinks it need so this will not necessarily include all availble units on the system. To see every unit file that is available in the systemd paths you can use the **list-unit-files** command instead:

```
$ systemctl list-unit-files
```

Units are representations of resources that systemd knows about. Because systemd has not necessarily read all of the unit definitions in this view it only presents information about hte files themselves. The output of this command shows two columns, the UNIT FILE and the STATE. The STATE column will usually be "enabled","disabled","static" or "masked". For this command static means that the unit file doesn't contain an "install" section which is necessary to enable a service. A unit that has a state of static can't be enable will run a one-off action or is only used as a dependency of another unit. We will cover what "masked" means later.

Unit management

So far we have been working with services an displaying information about the unit files that systemd knows about. With some additional commands we can get more specific information about units.

Displaying a unit file

To display the unit file that systemd has loaded into it's memory you can use the **cat** command (which was added in systemd version 209). To see the unit file of the atd scheduling daemon you can use to following command:

```
$ systemctl cat atd.service

[Unit]
Description=ATD daemon

[Service]
Type=forking
ExecStart=/usr/bin/atd

[Install]
WantedBy=multi-user.target
```

The output of this command is the unit file as it's known by the current systemd process. This is important to know if you've recently modified the unit files or if you're overriding certain options.

Displaying dependencies

If we want to see the dependency tree of a service we can use the **list-dependencies** command:

```
$ systemctl list-dependencies sshd.service
```

This command wil show a hierarchical view of the dependencies that must be dealt with in order to start the unit. Dependencies, in this context, are the units that are required or wanted by the units above it.

```
sshd.service
|-system.slice
`-basic.target
  |-microcode.service
  |-rhel-autorelabel-mark.service
  |-rhel-autorelabel.service
  |-rhel-configure.service
  |-rhel-dmesg.service
  |-rhel-loadmodules.service
  |-paths.target
  |-slices.target

. . .
```

Recursive dependencies are only displayed for target units which indicate system states. To list all dependencies you can include the `--all` flag.

To get the reverse dependencies you can add the `--reverse` flag to the command. Other useful flags are the `--before` and `--after` flags which can be used to show units to depend on the specified unit to start before or after them respectivly.

Checking unit properties

To get the low-level properties of a unit you can use the **show**. This will display a list of properties in a key=value format:

```
$ systemctl show sshd.service

Id=sshd.service
Names=sshd.service
Requires=basic.target
Wants=system.slice
WantedBy=multi-user.target
Conflicts=shutdown.target
Before=shutdown.target multi-user.target
After=syslog.target network.target auditd.service systemd-journald.socket basic.target ↩
    system.slice
Description=OpenSSH server daemon

. . .
```

To get a single property you can pass the `-p` flag with the property name. For example, to see the conflicts that the sshd.service unit has you can use the following command:

```
$ systemctl show sshd.service -p Conflicts

Conflicts=shutdown.target
```

Masking and unmasking units

In the service management section we showed how to stop or disable a service, but systemd also has the ability to mark a unit as completely unstartable. To do this it creates a symbolic link to `/dev/null` which is called masking a unit. To do this you can use the **mask** command:

```
$ systemctl mask nginx.service
```

This wil prevent the Nginx service from being start manually or automatically for as long as it's masked. If you check with the **list-unit-files** command you will see that the service is now listed as masked:

```
$ systemctl list-unit-files

. . .

kmod-static-nodes.service              static
ldconfig.service                       static
mandb.service                          static
messagebus.service                     static
nginx.service                          masked
quotaon.service                        static
rc-local.service                       static
rdisc.service                          disabled
rescue.service                         static

. . .
```

If you try to start the service you will see a message like this:

```
$ systemctl start nginx.service

Failed to start nginx.service: Unit nginx.service is masked.
```

To unmask a service you can use the **unmask** command:

```
$ systemctl unmask nginx.service
```

This will return the unit to it's previous start allowing the service to be started or enabled.

Editing unit files

The systemctl command also has the possibility to edit unit files if you need to make adjustments. This functionality was added in systemd version 218.

The **edit** will open a unit file snipper by default:

```
$ systemctl edit nginx.service
```

This will be a blank file that can be used to override or add properties to the unit definition. A directory will be created within the `/etc/systemd/system` directory which will have the name of the unit with .d appended. For example, for the nginx.service, a directory callend nginx.service.d will be created.

If you want to edit the full unit file, instead of adding a snippet, you can pass the `--full` flag:

```
$ systemctl edit --full nginx.service
```

This will load the current unit file into the editor where you can modify it. When the editor exits the changes will be written to `/etc/systemd/system`. This new file will take precedence over the system unit definition (usually found somewhere in `/lib/systemd/system`).

To remove any additions you have meid you can either remove the units .d configuration directory or the modiefied service file from `/etc/systemd/system`. For instance, to remove a snipper, you can type:

```
$ rm -r /etc/systemd/system/nginx.service.d
```

To remove a full modified unit file you can type:

```
$ rm /etc/systemd/system/nginx.service
```

After remove thile file or directory you should reload the systemd process so that it no longer attempts to reference the file and reverts back to using the system copies. You can do this by using the **daemon-reload** command:

```
$ systemctl daemon-reload
```

Adjusting the system start (runlevel) with targets

Targets are special unit files that describe a system state. Like other units the files that define targets can be identified by their suffix which in this case is **.target**. Targets don't do much themselves but ar used to group other units together.

These targets can be used to bring the system to a certain state, much like other init systems use runlevels. They are used as a reference for when certain functions are available allowing you to specify the desired state instead of the individual units needed to produce the same state.

For instance, there is a **swap.target** which is used to indicate that swap is ready for use. Units that are part of this process can sync with this target by indicating in their configuration files that they are wanted by or required by the **swap.target**. Unit that require swap to be available can specify this condition by using the wants, requires, and after properties to indicate the natur of their relationship.

Getting and setting the default target

Systemd has a default target that is used when booting the system. Satisfying the cascade of dependencies from that target will bring the system into the desired state. To get the default target of your system you can use the **get-default** command:

```
$ sytemctl get-default

multi-user.target
```

If you want to set another target as the default you can use the **set-default** command. For example, if you want to use the graphical desktop as default you can change this with the following command:

```
$ systemctl set-default graphical.target
```

Listing available targets

You can get a list of the available targets using the **list-unit-files** command in combination with the `--type=target` filter:

```
$ systemctl list-unit-files --type=target
```

Unlike with runlevels it's possible to have multiple targets active at the same time. An active target indicates that systemd has attempted to start all of the units tied to the target and has not tried to tear them down again. To see all active targets use the following command:

```
$ systemctl list-units --type=target
```

Isolating targets

It's possible to start all the units that are associated with a target and to stop all units that are not part of the dependency tree. The command we can use for this is the **isolate** command. This is similar to changing the runlevel in other init systems.

For example, if you are working in a graphical environment with graphical.target active, you can shutdown the graphical system by isolating the multi-user.target. Since multi-user.target is a dependency of graphical.target and not the other way around, the isolate command will stop all the graphical units.

You may wish to take a look at the dependencies of the target you're isolating to make sure you don't stop any services that are vital to you:

```
$ systemctl list-dependencies multi-user.target
```

When you're satisfied with the units that will be kept alive you can isolate the target:

```
$ systemctl isolate multi-user.target
```

Using shortcuts for important events

Some targets are defined for important events like powering off or rebooting. However **systemctl** also has some shortcuts that add a bit of additional functionality.

For instance, to put the system into rescue (single-user in System V init terms) mode you can just use the **rescue** instead of **isolate rescue.target**:

```
$ systemctl rescue
```

This command will also provide the additional functionallity of alerting all logged in users about the event in comparison with the isolate command. To halt the system you can use the **halt** command:

```
$ systemctl halt
```

To initiate a full shutdown you can use the **poweroff** command:

```
$ systemctl poweroff
```

To reboot the system you can use the **reboot** command:

```
$ systemctl reboot
```

Not that most systems will link the shorter, more conventional, commands for these operations so they will work properly with systemd. For example, to reboot a system you can usually type:

```
$ reboot
```

Questions and answers

System Maintenance

1. *Which GNU tar parameter supports processing a bzip2 compressed file?*

   The parameter **j** is used for processing bzip2 compressed files. Uncompress tarballs [145]

2. *What is most often the reason for a **configure** command to exit prematurely?*

   In case a required dependency is missing. However, in case only an optional dependency is missing, it will just disable compilation of that particular dependency. Check for optional and mandatory dependencies [146]

3. *According to GNU standards, an application will install by default into which directory?*

   Without optional arguments passed to the configure or the make install commands, the application will install into the `/usr/local/` directory. Default install directory [146]

4. *In general you would rather back up more than less. Which filesystems are major exceptions and can be skipped all together?*

   The `/proc` and `/sys` filesystems contain information about the state of the kernel and can therefore be skipped. What not to back-up [148]

5. *How often should you back up your system?*

   The frequency of your backup should be in pace with the amount of new data you create within a given timespan. If it is a lot, or very hard to recreate, consider a brief interval between backups. When to back-up? [148]

6. *Why is network attached storage not preferred for long term backup and archives?*

   Long term backups are hardly ever used and, if so, no instant recovery is expected. Knowing this, power costs should be taken into consideration as these systems are always up and running. Networked storage as backup medium [150]

7. *Why would **rsync** be a backup utility to be considered if bandwidth between the local and the remote destination is low?*

   rsync optimizes the use of bandwith by not only compressing traffic, but also to consider not copying files if not needed and to do this for parts of large files as well. Keeping required bandwith low [150]

8. *Why would you use* AMANDA *(Advanced Maryland Automatic Network Disk Archiver) when backing up a large number of systems running different versions of Linux and Unix?*

   AMANDA uses native utilities and formats and will most likely support multiple versions of Linux and Unix and be able to allow moving backup files across systems. Backup solutions [151]

9. *Does **ping**ing a system furnish enough information in order to decide whether a system is on-line or not?*

   No, as a firewall might be blocking ICMP echo requests. The same could also be blocking the replies. Ping can be blocked [133]

10. *When will a message within the file named `/etc/issue` be presented to the user?*

    Before authentication occurs in case the command **login** is used for logging in. Most clients for logging in will not use **login** in the end, **telnet** , however, does. Presenting /etc/issue [152]

Chapter 7

Domain Name Server (207)

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

Objective 207.1; Basic DNS server configuration (3 points) Candidates should be able to configure BIND to function as a caching-only DNS server. This objective includes the ability to convert older BIND configuration files to newer format, managing a running server and configuring logging.

Objective 207.2; Create and maintain DNS Zones (3 points) Candidates 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 records, adding hosts in zones and adding zones to the DNS. A candidate should also be able to delegate zones to another DNS server.

Objective 207.3; Securing a DNS server (2 points) Candidates should be able to configure a DNS server to run as a non-root user and run in a chroot jail. This objective includes secure exchange of data between DNS servers.

Note
The following chapters regarding DNS still mention the upgrade from BIND 8 to BIND 9. The LPIC-2 exam focusses on BIND 9. For a brief moment in history, there has been a BIND 10. But that got sold and re- branded. BIND 9 is what you should focus on when studying for the LPIC-2 exam.

Basic DNS server configuration (207.1)

Resources and further reading: Albitz01, DNShowto, BINDfaq, BIND at Wikipedia.org. RIPE: BIND 10.

Candidates should be able to configure BIND to function as an authoritive and as a recursive, caching-only DNS server. This objective includes the ability to manage a running server and configure logging.

Key Knowledge Areas

- BIND 9.x configuration files, terms and utilities.
- Defining the location of the BIND zone files in BIND configuration files.
- Reloading modified configuration and zone file.
- Awareness of dnsmasq, djbdns and PowerDNS as alternate name servers.

Terms and Utilities

- `/etc/named.conf`
- `/var/named/`
- **/usr/sbin/rndc**
- **/usr/sbin/named-checkconf**
- **kill**
- **dig**
- **host**

Name-server components in BIND

Name servers like BIND (Berkeley Internet Name Domain system) are part of a worldwide DNS system that resolves machine names to IP addresses.

In the early days of the Internet, host name to IP address mappings were maintained by the Network Information Center (NIC) in a single file called `HOSTS.TXT`. This file was then distributed by FTP to other Internet connected hosts.

Due to the vast amount of hosts being connected to the Internet over time, another name resolving system was developed known as Domain Names. This system incorporated design goals like distributed updates and local caching, to increase resolving performance. Because of these features, every nameserver needs to be specifically configured for it's purpose. The following terms are apparent when configuring nameserver software like BIND:

Zones are the equivalent of domains. Zone configuration files consist of hostnames and IP address information. *Nameserver* software responds to requests on port `53`, and translates DNS (host- or domain-)names to IP addresses. It can also translate IP addresses into DNS names, this is called a "reverse DNS lookup" (rDNS). In order for rDNS to work, a so called pointer DNS record (PTR record) has to exist for the host being queried.

We distinguish *authoritive* nameservers, *recursive* nameservers and so called *resolvers*. The authoritive nameserver for a zone is the nameserver which administrates the zone configuration. It is therefore sometimes also referred to as the zone *master*. A recursive nameserver is a nameserver that resolves zones for which it is not authoritive for at other nameservers. The resolver is the part of the nameserver and DNS client software which performs the actual queries. In general, these are libraries as part of DNS software.

Table Major BIND components lists the most relevant parts of BIND software on a system. Note that directories may vary across distributions.

Component	Description
Program `/usr/sbin/named`	the real name server
Program `/usr/sbin/rndc`	name daemon control program
Program `/usr/sbin/named-checkconf`	program to check named.conf file for errors
File `named.conf`	BIND configuration file
Program `/etc/init.d/bind`	distribution specific startfile
Directory `/var/named`	working directory for `named`

Table 7.1: Major BIND components

Resolving is controlled by the file `nsswitch.conf` which is mentioned in Chapter 10.

BIND components will be discussed below.

The `named.conf` file

The file `named.conf` is the main configuration file of BIND. It is the first configuration file read by **named**, the DNS name daemon.

Note
BIND 8 configuration files should work with BIND 9, although some modifications might be necessary. Some options do not work with BIND 9. BIND 9 offers new areas of configuration, please consult the BIND documentation and manpages when upgradading from BIND 8 to BIND 9.

Note
BIND 4 configuration files can be converted to BIND 9 format using a script called `named-bootconf.sh`

Location of `named.conf`

According to LPI the location of the file `named.conf` is in the `/etc` directory. 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* name server

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 that this is the *only* task a *caching-only* name server performs. It does not serve out zones, except for a few internal ones.

This is an example of a *caching-only* `named.conf` file. The version below is taken from the Debian bind package (some comments removed).

```
options {
        directory "/var/named";

        // 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.

Syntax

The `named.conf` file contains *statements* that start with a keyword plus an opening curly brace "{" and end with a closing curly brace "}". A statement may contain other statements. The `forwarders` statement is an example of this. A statement may also contain IP addresses or the `file` word followed by a filename. These simple statements *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`. However, it is a comment sign in BIND zone files, like the file `/etc/bind/db.local` from the `named.conf` example above. An example BIND zone file can be found in Section 7.2.3.1

The `options` statement

Of the many possible entries (see named.conf (5)) inside an `options` statement, only `directory`, `forwarders`, `forward`, `version` and `dialup` will be discussed below.

Note
There can be only *one* `options` statement in a `named.conf` file.

NOTABLE KEYWORDS IN `NAMED.CONF`

`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 can be found in `/etc/bind`. All zone files can also be found in the latter 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 IP 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 the same the Service Provider uses.

forward The `forward` works only when `forwarders` are specified.

Two values can be specified: `forward first;` (default) and `forward only;`. With `forward first`, the query is sent first to the specified name-server IP addresses and if this fails it should perform lookups elsewhere. With `forward only`, queries are limited only to the specified name-server IP 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 @ns12.zoneedit.com version.bind chaos txt

; <<>> DiG 9.8.3-P1 <<>> @ns12.zoneedit.com version.bind chaos txt
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59790
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;version.bind.      CH  TXT

;; ANSWER SECTION:
VERSION.BIND.   0 CH  TXT "8.4.X"

;; Query time: 169 msec
;; SERVER: 209.62.64.46#53(209.62.64.46)
;; WHEN: Tue Jun 25 11:38:48 2013
;; MSG SIZE  rcvd: 60
```

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";
```

or

```
version none;
```

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. Examples of unwanted actions are: sending heartbeat packets, zone transfers with a nameserver on the other side of the firewall.

The following example, also inside the `options` part, stops external zone maintenance:

```
heartbeat-interval 0;
dialup yes; // NOTE: This actually *stops* dialups!
```

Many more options can be placed inside the options block. Refer to the manual pages for details.

Depending on the distribution used, a seperate `bind.conf.options` file might be used which holds all the options for the BIND configuration. The main configuration file `named.conf` has to include this separate file though, which can be accomplished by adding the following line to `named.conf`:

```
include "/etc/bind/named.conf.options";
```

Other separate configuration files like `named.conf.log` or `named.conf.default-zones` may be nested this way as well.

The `logging` statement

The BIND (version 8 and 9) logging system is too elaborate to discuss in detail here. An important difference between the two has to do with parsing the log configuration. BIND 8 used to parse the `logging` statement and start the logging configuration right away. BIND 9 only establishes the logging configuration *after* the entire configuration file has been parsed. While starting up, the server sends all logging messages regarding syntax errors in the configuration file to the default channels. These errors may be redirected to standard error output if the `-g` option has been given during startup.

The distinction between *categories* and *channels* is an important part of logging.

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` can be referred to using the "@" symbol 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 Section 7.2.

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 lists ways to control the named name server.

Method	See
The rndc program	Section 7.1.6
Sending signals	Section 7.1.8
Using a start/stop script	Section 7.1.9

Table 7.2: Controlling named

The rndc program

The **rndc** (Remote Name Daemon Control) program can be used to control the named name server daemon, locally as well as remotely. It requires a /etc/rndc.key file which contains a key.

```
key "rndc-key" {
    algorithm hmac-md5;
    secret "tyZqsLtPHCNna5SFBLT0Eg==";
};

options {
    default-key "rndc-key";
    default-server 127.0.0.1;
    default-port 953;
};
```

The name server configuration file /etc/named.conf needs to contain the same key to allow a host to control the name server. The relevant part of that file is shown below.

```
key "rndc-key" {
    algorithm hmac-md5;
    secret "tyZqsLtPHCNna5SFBLT0Eg==";
};

controls {
    inet 127.0.0.1 port 953
        allow { 127.0.0.1; } keys { "rndc-key"; };
};
```

The secret itself will never be transmitted over the network. Both ends calculate a hash using the algorithm declared after the algorithm keyword and compare hashes.

Depending on the distribution used, the configuration files might also be stored in /etc/bind/*. The rndc.key file should be owned by root:bind and have a mode of 640. The main bind configuration file, named.conf should have a line that includes the keyfile. Running the program **rndc-confgen** will create a key in case none is available on the system.

A command to the name server can be given as a parameter to rndc, e.g.: **rndc reload**. This will request the name server to reload its configuration and zone files. All commands specified in this way are understood by the name daemon. The help command presents a list of commands understood by the name server.

While not discussed here **rndc** may be used to manage several name servers remotely. Consult the man pages and the "BIND 9 Administrator Reference Manual" for more information on rndc and BIND.

The named-checkconf utility

named-checkconf is a very useful utility that checks the `named.conf` file for errors. If the `named.conf` file is located in the regular `/etc/named.conf` location on your distribution you only have to type in the command to check the file.

```
# named-checkconf
```

The location of the named.conf file can be different however (depending on the distribution you are using). In some cases for example the file is located at `/etc/bind/named.conf`. The **named-checkconf** utility wil not automatically recognize locations other than `/etc/named.conf` in these cases you will have to include path and filename after the command.

```
# named-checkconf /etc/bind/named.conf
```

When the prompt returns without giving any messages it means that **named-checkconf** didn't find anything wrong with it. The example below will show what happens when it dit find something wrong. In this case I made an error by forgetting to add the letter **i** on an **include** statement.

```
[root@localhost etc]# named-checkconf
/etc/named.conf:56: unknown option 'nclude'
```

The **named-checkconf** utility will only check the `named.conf` file. Other configuration files called from within the `named.conf` file using for example the **include** statement will not be checked automatically. It it possible to check them manually by adding their path and file name when executing the `named.checkconf` utility.

Sending signals to named

It is possible to send signals to the `named` process to control its behaviour. 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 or `/etc/init.d/named` in Red Hat.

Note
Red Hat (based) systems have the **service** command which can be used instead. **service** uses the same set of parameters, so you might, for example, say:

```
# service named reload
```

Table Table 7.3 lists parameters which a current version of `/etc/init.d/bind` accepts.

dnsmasq

dnsmasq is both a lightweight DNS forwarder and DHCP server. **dnsmasq** supports static and dynamic DHCP leases and supports BOOTP/TFTP/PXE network boot protocols.

```
$ apt-cache search dnsmasq
dnsmasq - Small caching DNS proxy and DHCP/TFTP server
dnsmasq-base - Small caching DNS proxy and DHCP/TFTP server
dnsmasq-utils - Utilities for manipulating DHCP leases
```

Parameter	Description
start	starts named
stop	stops named
restart	stops and restarts named
reload	reloads configuration
force-reload	same as restart

Table 7.3: /etc/init.d/bind parameters

djbdns

djbdns - Daniel J. Bernstein DNS - was build due to frustrations with repeated BIND security holes. Besides holding a DNS cache, DNS server and DNS client djbdns also includes several DNS debugging tools. The source code was released into the public domain in 2007. There have been several forks, one of which is dbndns, the fork of the Debian Project.

PowerDNS

PowerDNS is a Dutch supplier of DNS software and services. The PowerDNS software is open source (GPL), and comes packaged with many distributions as pdns, powerdns-server or pdns-server. The system allows multiple backends to allow access to DNS configuration data, including a simple backend that accepts BIND style files.

```
$ apt-cache search pdns
pdns-backend-geo - geo backend for PowerDNS
pdns-backend-ldap - LDAP backend for PowerDNS
pdns-backend-lua - lua backend for PowerDNS
pdns-backend-mysql - generic MySQL backend for PowerDNS
pdns-backend-pgsql - generic PostgreSQL backend for PowerDNS
pdns-backend-pipe - pipe/coprocess backend for PowerDNS
pdns-backend-sqlite - sqlite backend for PowerDNS
pdns-backend-sqlite3 - sqlite backend for PowerDNS
pdns-server - extremely powerful and versatile nameserver
pdns-server-dbg - debugging symbols for PowerDNS
pdns-recursor - PowerDNS recursor
pdns-recursor-dbg - debugging symbols for PowerDNS recursor
pdnsd - Proxy DNS Server
```

The dig and host utilities

The Internet Systems Consortium (ICS) has deprecated **nslookup** in favor of **host** and **dig**. However, **nslookup** is still widely used due to longevity of older Unix releases. It remains part of most Linux distributions too.

Both **dig** and **host** commands can be used to query nameservers, it's a matter of preference which one to use for which occasion. **dig** has far more options and provides a more elaborate output by default. The *help* for both commands should give some insights in the differences:

```
$ host
Usage: host [-aCdlriTwv] [-c class] [-N ndots] [-t type] [-W time]
            [-R number] [-m flag] hostname [server]
       -a is equivalent to -v -t ANY
       -c specifies query class for non-IN data
       -C compares SOA records on authoritative nameservers
       -d is equivalent to -v
       -l lists all hosts in a domain, using AXFR
       -i IP6.INT reverse lookups
       -N changes the number of dots allowed before root lookup is done
       -r disables recursive processing
       -R specifies number of retries for UDP packets
```

```
        -s a SERVFAIL response should stop query
        -t specifies the query type
        -T enables TCP/IP mode
        -v enables verbose output
        -w specifies to wait forever for a reply
        -W specifies how long to wait for a reply
        -4 use IPv4 query transport only
        -6 use IPv6 query transport only
        -m set memory debugging flag (trace|record|usage)
```

```
$ dig -h
Usage:  dig [@global-server] [domain] [q-type] [q-class] {q-opt}
            {global-d-opt} host [@local-server] {local-d-opt}
            [ host [@local-server] {local-d-opt} [...]]
Where:  domain    is in the Domain Name System
        q-class  is one of (in,hs,ch,...) [default: in]
        q-type   is one of (a,any,mx,ns,soa,hinfo,axfr,txt,...) [default:a]
                 (Use ixfr=version for type ixfr)
        q-opt    is one of:
                 -x dot-notation     (shortcut for reverse lookups)
                 -i                  (use IP6.INT for IPv6 reverse lookups)
                 -f filename         (batch mode)
                 -b address[#port]   (bind to source address/port)
                 -p port             (specify port number)
                 -q name             (specify query name)
                 -t type             (specify query type)
                 -c class            (specify query class)
                 -k keyfile          (specify tsig key file)
                 -y [hmac:]name:key  (specify named base64 tsig key)
                 -4                  (use IPv4 query transport only)
                 -6                  (use IPv6 query transport only)
                 -m                  (enable memory usage debugging)
        d-opt    is of the form +keyword[=value], where keyword is:
                 +[no]vc             (TCP mode)
                 +[no]tcp            (TCP mode, alternate syntax)
                 +time=###           (Set query timeout) [5]
                 +tries=###          (Set number of UDP attempts) [3]
                 +retry=###          (Set number of UDP retries) [2]
                 +domain=###         (Set default domainname)
                 +bufsize=###        (Set EDNS0 Max UDP packet size)
                 +ndots=###          (Set NDOTS value)
                 +edns=###           (Set EDNS version)
                 +[no]search         (Set whether to use searchlist)
                 +[no]showsearch     (Search with intermediate results)
                 +[no]defname        (Ditto)
                 +[no]recurse        (Recursive mode)
                 +[no]ignore         (Don't revert to TCP for TC responses.)
                 +[no]fail           (Don't try next server on SERVFAIL)
                 +[no]besteffort     (Try to parse even illegal messages)
                 +[no]aaonly         (Set AA flag in query (+[no]aaflag))
                 +[no]adflag         (Set AD flag in query)
                 +[no]cdflag         (Set CD flag in query)
                 +[no]cl             (Control display of class in records)
                 +[no]cmd            (Control display of command line)
                 +[no]comments       (Control display of comment lines)
                 +[no]question       (Control display of question)
                 +[no]answer         (Control display of answer)
                 +[no]authority      (Control display of authority)
                 +[no]additional     (Control display of additional)
                 +[no]stats          (Control display of statistics)
                 +[no]short          (Disable everything except short
                                      form of answer)
```

```
                  +[no]ttlid          (Control display of ttls in records)
                  +[no]all            (Set or clear all display flags)
                  +[no]qr             (Print question before sending)
                  +[no]nssearch       (Search all authoritative nameservers)
                  +[no]identify       (ID responders in short answers)
                  +[no]trace          (Trace delegation down from root)
                  +[no]dnssec         (Request DNSSEC records)
                  +[no]nsid           (Request Name Server ID)
                  +[no]sigchase       (Chase DNSSEC signatures)
                  +trusted-key=####   (Trusted Key when chasing DNSSEC sigs)
                  +[no]topdown        (Do DNSSEC validation top down mode)
                  +[no]multiline      (Print records in an expanded format)
                  +[no]onesoa         (AXFR prints only one soa record)
         global d-opts and servers (before host name) affect all queries.
         local d-opts and servers (after host name) affect only that lookup.
         -h                           (print help and exit)
         -v                           (print version and exit)
```

As demonstrated, the **dig** command provides the broader range of options. Without options though, the provided information is quite similar:

```
$ host zonetransfer.me
zonetransfer.me has address 217.147.180.162
zonetransfer.me mail is handled by 20 ASPMX2.GOOGLEMAIL.COM.
zonetransfer.me mail is handled by 20 ASPMX3.GOOGLEMAIL.COM.
zonetransfer.me mail is handled by 20 ASPMX4.GOOGLEMAIL.COM.
zonetransfer.me mail is handled by 20 ASPMX5.GOOGLEMAIL.COM.
zonetransfer.me mail is handled by 0 ASPMX.L.GOOGLE.COM.
zonetransfer.me mail is handled by 10 ALT1.ASPMX.L.GOOGLE.COM.
zonetransfer.me mail is handled by 10 ALT2.ASPMX.L.GOOGLE.COM.
```

```
$ dig zonetransfer.me

; <<>> DiG 9.8.4-rpz2+rl005.12-P1 <<>> zonetransfer.me
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31395
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1

;; QUESTION SECTION:
;zonetransfer.me.   IN  A

;; ANSWER SECTION:
zonetransfer.me.  7193  IN  A 217.147.180.162

;; AUTHORITY SECTION:
zonetransfer.me.  7193  IN  NS  ns12.zoneedit.com.
zonetransfer.me.  7193  IN  NS  ns16.zoneedit.com.

;; ADDITIONAL SECTION:
ns12.zoneedit.com.  1077  IN  A 209.62.64.46

;; Query time: 6 msec
;; SERVER: 213.154.248.156#53(213.154.248.156)
;; WHEN: Thu Jun 27 07:30:36 2013
;; MSG SIZE  rcvd: 115
```

$ **man dig**

```
NAME
       dig - DNS lookup utility
```

```
SYNOPSIS
       dig [@server] [-b address] [-c class] [-f filename] [-k filename] [-m] [-p port#]
           [-q name] [-t type] [-x addr] [-y [hmac:]name:key] [-4] [-6] [name] [type]
           [class] [queryopt...]
```

An important option is `-t` to query for example only the MX or NS records. This option works for **dig** and **host**. (Look at the man pages of both commands for the explanation of the other options.)

```
$ dig snow.nl

; <<>> DiG 9.8.3-P1 <<>> snow.nl
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60605
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;snow.nl.     IN  A

;; ANSWER SECTION:
snow.nl.    299 IN  A 213.154.248.202

;; Query time: 52 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Oct 20 09:55:13 2015
;; MSG SIZE  rcvd: 41
```

```
$ dig -t NS snow.nl

; <<>> DiG 9.8.3-P1 <<>> -t NS snow.nl
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26099
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;snow.nl.     IN  NS

;; ANSWER SECTION:
snow.nl.    20361 IN  NS  ns1.transip.nl.
snow.nl.    20361 IN  NS  ns2.transip.eu.
snow.nl.    20361 IN  NS  ns0.transip.net.

;; Query time: 59 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Oct 20 10:00:24 2015
;; MSG SIZE  rcvd: 108
```

```
$ dig -t MX snow.nl

; <<>> DiG 9.8.3-P1 <<>> -t MX snow.nl
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36671
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;snow.nl.     IN  MX

;; ANSWER SECTION:
```

```
snow.nl.    299 IN  MX  10 mc.snow.nl.
snow.nl.    299 IN  MX  20 mx1.snow.nl.

;; Query time: 61 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Oct 20 10:02:01 2015
;; MSG SIZE  rcvd: 64
```

Create and maintain DNS zones (207.2)

Resources and further reading: Albitz01, DNShowto, BINDfaq.

Candidates should be able to create a zone file for a forward or reverse zone and hints for root level servers. This objective includes setting appropriate values for records, adding hosts in zones and adding zones to the DNS. A candidate should also be able to delegate zones to another DNS server.

Key Knowledge Areas

- BIND 9 configuration files, terms and utilities
- Utilities to request information from the DNS server
- Layout, content and file location of the BIND zone files
- Various methods to add a new host in the zone files, including reverse zones

Terms and Utilities

- `/var/named/*`
- zone file syntax
- resource record formats
- **named-checkzone**
- **named-compilezone**
- **masterfile-format**
- **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 Section 7.2.3.4 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 Section 7.2.5.1.

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 the authoritative name server to query about this domain. The SOA record should contain the administrator contact address as well as a time-out setting for slave nameservers. Declaring a SOA record serves 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.

The SOA record is mandatory for every DNS zone file, and should be the first specified Resource Record (RR) of a zone file as well.

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
             3600 )  ; 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. From rfc4343: *"As described in [STD13] and [RFC2929], DNS has an additional axis for data location called CLASS. The only CLASS in global use at this time is the "IN" (Internet) CLASS."*

`SOA` start of authority - that's what this section is about.

`lion.example.org.` the name of the machine that has the master (see Section 7.2.4) 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 on 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 (3600s)`

The A resource record

The A record 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 should have a corresponding PTR record. This is described in Section 7.2.3.5.

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 (Canonical Name) record specifies another name for a host with an A record. BIND 8 used to allow multiple CNAME records, by accepting an option called multiple-cnames. From BIND 9.2 onward though, the CNAME rules are strictly enforced in compliance to the DNS standards. 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 the 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.

Note
There should be at least two independent name servers for each domain. Independent means connected to separate networks, separate power lines, etc. See Section 7.2.4 below.

The MX resource record

The MX (Mail Exchanger) record system is used by the mail transfer system, e.g., the sendmail and postfix daemons. Multiple MX records may be provided for each domain. 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 to. If that host is down, another host with a lower priority (and therefore a higher number) 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.

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` in the `example.org` zone file, e.g.:

```
; accept mail for the domain too
example.org.  IN  MX   0   lion.example.org.
example.org.  IN  MX  10   cat.example.org.
```

Mail addressed to `example.org` will only be accepted by the MTA on `lion.example.org` host if the MTA is configured for this.

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 among other things used to check whether a host name belongs to a specific 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
                 3600 )  ; 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.
```

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`).

IPv6 records

The IPv6 address format

IPv6 addresses are 128 bit addresses rather than IPv4's 32 bits. They are notated as 8 groups of 16-bit values, written in 4 hexadecimal numbers per part, separated by a colon. For the sake of readability leading zero's in every part may be omitted, and parts consisting of zero's only may be completely omitted. The latter may only be done once per address because multiple occurrences would create an ambiguous representation.

For example: `2001:0db8:0000:0000:0000:ff00:0042:8329` can be rewritten to `2001:db8::ff00:42:8329`.

The localhost (loopback) address `0:0:0:0:0:0:0:1` can be reduced to `::1`

The AAAA record

Where the `A` record is used for IPv4, the `AAAA` record is used for IPv6. It resolves a hostname to an IPv6 address in the same way as an `A` record does for IPv4.

```
lion   IN   AAAA    2001:db8::ff00:42:8329
```

Note
Note: Another format used to resolve IPv6 address records was the A6 record. While supported by the current BIND versions it is considered "historic" by RFC6563

The PTR record

For reverse resolution IPv6 address are represented in another format, with every hexadecimal digit separated by a dot. Our example above becomes: `2.0.0.1.0.d.b.8.0.0.0.0.0.0.0.0.0.0.0.0.f.f.0.0.0.0.4.2.8.3.2.9`

The IPv6 PTR record is that address in exact reverse order with the domain `ip6.arpa` appended. The above address becomes: `9.2.3.8.2.4.0.0.0.0.f.f.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa`

Part of that domain may be delegated, just as IPv4 addresses. In `named.conf` this could be the corresponding entry:

```
zone "0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa" IN {
    type master;
    file "/etc/bind/exampleorg-ip6.rev";
};
```

The corresponding file should look like the following file:

```
$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.

9.2.3.8.2.4.0.0.0.0.f.f.0.0.0.0      IN   PTR     lion.example.org.
```

As both IPv4 and IPv6 PTR records are in different zones (i.e. `in-addr.arpa` and `ip6.arpa`) they should be defined in separate zone files.

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 Section 7.2.5.1). That is, they both give the same answers about the zone.

The data of a zone originates from 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`) that defines the `example.org` master.

```
zone "example.org" IN {
type master;
file "/etc/bind/exampleorg.zone";
};
```

Of course, the `exampleorg.zone` file must 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 (e.g. when 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. In other words: the `NS` record for a zone should specify all slaves for that 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 `db.example.org` 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

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. In other words, the DNS server hosting the stub zone is only a source of information on the authoritative name servers. This server must have network access to the remote DNS server in order to copy the authoritative name server information.

The purpose of stub zones is two fold:

- *Keep delegated zone information current.* By updating a stub zone for one of its child zones regularly, the DNS server that hosts the stub zone will maintain a current list of authoritative DNS servers for the child zone.
- *Improve name resolution.* Stub zones make it possible for a DNS server to perform name resolution using the stub zone's list of name servers, without having to use forwarding or root hints.

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. This is described in the next section.

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).

Checking zone files for syntax errors

After creating or making changes to a zone file it is a good idea to check them for syntax errors. This can be done with the **named-checkzone** command. The way to do this is by entering the domain name after the command followed by the name of the zone file. See below for an example.

```
# named-checkzone test.com /var/named/test.com.zone
zone test.com/IN: loaded serial 0
OK
```

In this example I didn't put a dot at the end of the domain name but you can choose to do so. Both ways are correct and will give the same answer.

As you can see in the example above, the utility gives an "OK" as a response this means the file is clear of any syntax errors. You maybe also want to consider checking the man page for **named-checkzone** for additional options.

From version 9.9 of the BIND software secondary server zone files are by default saved in raw binary format instead of text format. Reading and checking these files can be a bit more challenging because of this. Luckily there is the tool **named-compilezone**. This tool is quite similair to the **named-checkzone** tool but makes it possible to change raw zone files to text format (and vice versa) so they are readable by normal human beings.

It is possible to change to change this behaviour and change the default format back to text format. This can be done by adding the **masterfile-format** option to the zone statement in the named.conf files on the secondary name servers that you administer. Example below shows the correct syntax for this.

```
masterfile-format text;
```

DNS Utilities

Four DNS utilities can help to resolve names and even debug a DNS zone. They are called **dig**, **host**, **nslookup** and **dnswalk**. The first 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** command 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. For example:

```
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. As stated before the use of **nslookup** is deprecated, and the commands **host** and **dig** should be used instead. Despite this recommendation, you should have some knowledge of its usage.

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 in the next section), 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.

DNSwalk

DNSwalk is a DNS debugger. Use with caution, since it tries to perform zone transfers while checking DNS databases for consistency and accuracy. Example:

```
$ dnswalk zoneedit.com.
Checking zoneedit.com.
Getting zone transfer of zoneedit.com. from ns2.zoneedit.com...done.
SOA=ns2.zoneedit.com  contact=soacontact.zoneedit.com
WARN: zoneedit.com A 64.85.73.107: no PTR record
WARN: golf.zoneedit.com A 69.72.176.186: no PTR record
WARN: zoneapp1.zoneedit.com A 64.85.73.104: no PTR record
WARN: dynamic1.zoneedit.com A 64.85.73.40: no PTR record
WARN: zoneapp2.zoneedit.com A 64.85.73.107: no PTR record
WARN: ezzi.zoneedit.com A 207.41.71.242: no PTR record
WARN: dynamic2.zoneedit.com A 64.85.73.40: no PTR record
WARN: legacyddns.zoneedit.com A 64.85.73.40: no PTR record
WARN: api2.zoneedit.com A 64.85.73.104: no PTR record
WARN: wfb.zoneedit.com A 69.72.142.98: no PTR record
WARN: new.zoneedit.com A 64.85.73.107: no PTR record
WARN: zebra.zoneedit.com A 69.72.240.114: no PTR record
WARN: api.zoneedit.com A 64.85.73.40: no PTR record
WARN: www.zoneedit.com A 64.85.73.107: no PTR record
WARN: newapi.zoneedit.com A 64.85.73.104: no PTR record
0 failures, 15 warnings, 0 errors.
```

Securing a DNS Server (207.3)

Resources and further reading: Albitz01, BINDfaq, DNShowto, Lindgreen01, ChrootBind9 Nijssen99, Choose the right DNS configuration, Tutorial on DANE and DNSSEC. LPIC2sybex2nd, Let's Encrypt TLSA. DANE at Wikipedia.org. DigiNotar at Wikipedia.org. RFC6698.

Candidates should be able to configure a DNS server to run as a non-root user and run in a chroot jail. This objective includes secure exchange of data between DNS servers.

Key Knowledge Areas

- BIND 9 configuration files
- Configuring BIND to run in a chroot jail
- Split configuration of BIND using the forwarders statement
- Configuring and using transaction signatures (TSIG)
- Awareness of DNSSEC and basic tools
- Awareness of DANE and related records

Terms and Utilities

- `/etc/named.conf`
- `/etc/passwd`
- DNSSEC
- **dnssec-keygen**
- **dnssec-signzone**

DNS Security Strategies

There are several strategies possible to make DNS more secure.

When a security bug is discovered in the BIND nameserver, it will often be fixed within a few hours. This results in a new BIND release. Hence, older versions of BIND may have security related bugs. BIND users should frequently visit the BIND source site (http://www.isc.org/downloads/bind/) to check for new updates. You may consider subscribing to a security announcement list; most Linux distributions have one. For instance, the Debian `debian-security-announce`. To subscribe to that list visit https://lists.debian.org.

Making information harder to get

Like all other programs, the BIND nameserver has features and security holes that can be misused. Criminal hackers (crackers) are well aware of this and keep track of versions and their security related issues. It is wise not to advertise your vulnerabilities so you may want to disable version number reporting in BIND.

Also, bulk nameserver data can be used to disclose details about your infrastructure. Hence restrictions can be installed to prevent a user from obtaining a large amount of nameserver data or at least to get it in a short timeframe. Another reason to do so is that nameservers are built to answer a large number of short queries in a short timeframe. Having to serve large blobs of data unexpectedly might disrupt services.

It is important to remember that hiding information is *security through obscurity*. Doing so, the easy way to get information has been blocked but there may be other ways to get it. *Security through obscurity* makes it harder, but not impossible to get the information. It does however introduce a longer timeframe in which you may be able to detect break-in attempts and act accordingly, for example by blocking the requestors IP address in your firewalls.

Hiding the version number

The command

```
dig @target chaos version.bind txt
```

will show the version of the BIND nameserver 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 nameserver data. 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`. ACL labels are used to simplify administration: you only need to update your ACLs in one place, after that each reference to a label will automatically point to the updated data.

A nameserver supports queries by resolvers and zone transfers by other nameservers.

normal queries Normal queries occur when a resolver needs data only on a very limited subset of the data. For example it wants to know which IP address is associated with a hostname. The `allow-query` statement controls from which hosts these queries are accepted.

zone transfers The internet name space is built on the idea of a hierarchy of domains. To find the IP address of a host, you need its fully qualified domain name (*FQDN*). It consists of a series of subdomains separated by dots, for example: `www.example.com`. The first section is the hostname, i.e. "www". Next is the "example" domain, which in turn is a subdivision of the "com" domain. Data on a single domain is called a "zone". Slave (backup) nameservers must be allowed to get the information from a master of the same zone. It is important to restrict these "zone transfers" to the servers that need the data only - no more.

Zone transfers are controlled by the `allow-transfer` statement. This statement may also be specified in the `zone` statement, thereby overriding the global `options allow-transfer` statement. Traditionally zone transfers can be initiated by anybody.

The `allow-query` statement can be used inside a `zone` statement or an `options` statement as well. It can contain either an acl label (like `trusted`), `none`, or one or more IP addresses or IP ranges. Use labels whenever you can: keeping track of permissions is much easier that way and it makes you less vulnerable to accidental oversights that may cause security holes.

Limiting zone transfers

Both **dig** and **host** can initiate a *zone transfer*. By default, both master and slave servers are allowed to send all zone information. An example using the **dig** command:

```
$ dig axfr @ns12.zoneedit.com zonetransfer.me

; <<> DiG 9.8.3-P1 <<>> axfr @ns12.zoneedit.com zonetransfer.me
; (1 server found)
;; global options: +cmd
zonetransfer.me.  7200  IN  SOA ns16.zoneedit.com. soacontact.zoneedit.com. \
  2013064418 2400 360 1209600 300
zonetransfer.me.  7200  IN  NS  ns16.zoneedit.com.
zonetransfer.me.  7200  IN  NS  ns12.zoneedit.com.
zonetransfer.me.  7200  IN  A 217.147.180.162
zonetransfer.me.  7200  IN  MX  0 ASPMX.L.GOOGLE.COM.
zonetransfer.me.  7200  IN  MX  10 ALT1.ASPMX.L.GOOGLE.COM.
zonetransfer.me.  7200  IN  MX  10 ALT2.ASPMX.L.GOOGLE.COM.
zonetransfer.me.  7200  IN  MX  20 ASPMX2.GOOGLEMAIL.COM.
zonetransfer.me.  7200  IN  MX  20 ASPMX3.GOOGLEMAIL.COM.
zonetransfer.me.  7200  IN  MX  20 ASPMX4.GOOGLEMAIL.COM.
zonetransfer.me.  7200  IN  MX  20 ASPMX5.GOOGLEMAIL.COM.
zonetransfer.me.  301 IN  TXT "Remember to call or email Pippa on +44 123 4567890 \
  or pippa@zonetransfer.me when making DNS changes"
zonetransfer.me.  301 IN  TXT "google-site-verification= ↩
    tyP28J7JAUHA9fw2sHXMgcCC0I6XBmmoVi04VlMewxA"
```

```
testing.zonetransfer.me. 301  IN  CNAME www.zonetransfer.me.
164.180.147.217.in-addr.arpa.zonetransfer.me. 7200 IN PTR www.zonetransfer.me.
ipv6actnow.org.zonetransfer.me. 7200 IN AAAA  2001:67c:2e8:11::c100:1332
asfdbauthdns.zonetransfer.me. 7900 IN AFSDB 1 asfdbbox.zonetransfer.me.
office.zonetransfer.me. 7200  IN  A 4.23.39.254
owa.zonetransfer.me.  7200  IN  A 207.46.197.32
info.zonetransfer.me. 7200  IN  TXT "ZoneTransfer.me service provided by Robin \
  Wood - robin@digininja.org. See www.digininja.org/projects/zonetransferme.php for more  ↩
      information."
asfdbbox.zonetransfer.me. 7200  IN  A 127.0.0.1
canberra_office.zonetransfer.me. 7200 IN A  202.14.81.230
asfdbvolume.zonetransfer.me. 7800 IN  AFSDB 1 asfdbbox.zonetransfer.me.
email.zonetransfer.me.  2222  IN  NAPTR 1 1 "" "E2U+email" "" email.zoneedit.com. ↩
    zonetransfer.me.
dzc.zonetransfer.me.  7200  IN  TXT "AbCdEfG"
dr.zonetransfer.me. 300 IN  LOC 53 20 56.558 N 1 38 33.526 W 0.00m 1m 10000m 10m
rp.zonetransfer.me. 321 IN  RP  robin.zonetransfer.me.zonetransfer.me. robinwood. ↩
    zonetransfer.me.
sip.zonetransfer.me.  3333  IN  NAPTR 2 3 "au" "E2U+sip" "!^.*$!sip:customer- ↩
    service@zonetransfer.me!".
alltcpportsopen.firewall.test.zonetransfer.me. 301 IN A 127.0.0.1
www.zonetransfer.me.  7200  IN  A 217.147.180.162
staging.zonetransfer.me. 7200 IN  CNAME www.sydneyoperahouse.com.
deadbeef.zonetransfer.me. 7201  IN  AAAA  dead:beaf::
robinwood.zonetransfer.me. 302  IN  TXT "Robin Wood"
vpn.zonetransfer.me.  4000  IN  A 174.36.59.154
_sip._tcp.zonetransfer.me. 14000 IN SRV 0 0 5060 www.zonetransfer.me.
dc_office.zonetransfer.me. 7200 IN  A 143.228.181.132
zonetransfer.me.  7200  IN  SOA ns16.zoneedit.com. soacontact.zoneedit.com. \
  2013064418 2400 360 1209600 300
;; Query time: 334 msec
;; SERVER: 209.62.64.46#53(209.62.64.46)
;; WHEN: Fri Jun 28 12:01:16 2013
;; XFR size: 37 records (messages 37, bytes 2673)
```

The previous example shows the complete information about `zonetransfer.me`. The command **host -l** does the same but formats its output differently:

```
$ host -l zonetransfer.me ns16.zoneedit.com
Using domain server:
Name: ns16.zoneedit.com
Address: 69.64.68.41#53
Aliases:

zonetransfer.me nameserver ns16.zoneedit.com.
zonetransfer.me nameserver ns12.zoneedit.com.
zonetransfer.me has address 217.147.180.162
164.180.147.217.in-addr.arpa.zonetransfer.me domain name pointer www.zonetransfer.me.
ipv6actnow.org.zonetransfer.me has IPv6 address 2001:67c:2e8:11::c100:1332
office.zonetransfer.me has address 4.23.39.254
owa.zonetransfer.me has address 207.46.197.32
asfdbbox.zonetransfer.me has address 127.0.0.1
canberra_office.zonetransfer.me has address 202.14.81.230
alltcpportsopen.firewall.test.zonetransfer.me has address 127.0.0.1
www.zonetransfer.me has address 217.147.180.162
deadbeef.zonetransfer.me has IPv6 address dead:beaf::
vpn.zonetransfer.me has address 174.36.59.154
dc_office.zonetransfer.me has address 143.228.181.132
```

Only validated slave nameservers should be allowed to issue a *zone transfer* request. This can be done by making adjustments to the configuration files (`named.conf`) of both the master and all slaves of a particular zone. For example:

On master nameservers On the master nameserver you should use the `allow-transfer` statement 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 the host `cat` in the example) can request a *zone transfer* from this master nameserver.

On slave nameservers On a *slave* nameserver you should never allow any zone transfers. This can be achieved by setting the `allow-transfer` clause to 'none':

```
zone "example.org" IN {
    type slave;
    // ....

    allow-transfer {
        none;
    };
};
```

Note
IMPORTANT: Don't forget to protect the *reverse zone* as well!

Limiting queries

Though strictly spoken any client should be allowed to query a nameserver, in practice you may want to limit queries to a range of hosts, for example the hosts in your own networks. This can be done as follows:

```
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

There are more security controls that can be used.

Spoofing is when a malicious nameserver answers a client request with false data. As nameserver data is extensively cached for performance reasons, cached data can get infected with the falsified data. *Spoofing* is quite difficult to achieve, but it is wise to take precautions against it. Many older spoofing methods have already been eliminated by patches in the software but BIND can be configured to further enhance spoofing protection.

Limiting effects of an intrusion

Even when you are prepared to implement DNS fixes as soon as they appear, it may be too late - even if by only a couple of hours: your system could be compromised anyway. Therefore it is advisable to take precautions to minimize the impact.

Running BIND with less privileges

In some distributions, BIND runs as `root`. If BIND is compromised, the attacker might have root access. This can be prevented by running BIND under a non-privileged user and group.

It might be tempting to employ the user `nobody` and group `nogroup`. But since many services already do this another security issue arises: these services may be able to communicate in one way or another.

Best practice is to create a special user, e.g. `named`, and a corresponding group (which could also be called `named`), and run the nameserver under this user/group combination.

Use the `-u` and `-g` options to run **named** as `named`/`named`:

```
named -u named -g named
```

On a Debian system the start line in `/etc/init.d/bind` would be:

```
start-stop-daemon ... --exec /usr/sbin/named -- -u named -g named
```

(For clarity other options were replaced by dots). The extra `--` tells the `start-stop-daemon` where its options end, all other options will be passed on to the **named** program.

On a Red Hat (or compatible) system, the startup file can be found in `/etc/rc.d/init.d/named`. On RHEL systems the service is already configured to run under the user 'named'. You can set additional options in `/etc/sysconfig/named`, for example you can add `-u named` there. To start the service on Red Hat (compatible) systems, use the **service** command:

```
# service named start
Generating /etc/rndc.key:                                  [  OK  ]
Starting named:                                            [  OK  ]
```

To make **named** start up in the proper runlevels on Red Hat (compatible) systems run **chkconfig named on**.

> **Note**
> Make sure the nameserver 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*. On startup the process will set a directory (e.g. `/var/cache/bind`) as its root directory. Since no process is ever capable of accessing anything above its set root, this prevents the named process access to anything outside `/var/cache/bind`.

Preparing a chroot jail

All files that BIND needs must be copied to the new root directory. Full details can be found here: ChrootBind9.

In short:

- You will need to copy the `/etc`, `/dev`, `/lib`, `/sbin` or `/usr/sbin`, and `/var/run` directories to the chroot.
- You'll probably need the `/dev/null` device under the new root. The next command will create it:

```
mknod -m 666 /var/cache/bind/dev/null c 1 3
```

- 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 Section 7.3.6.3.

  You may also need to copy `ld.so.cache` and `localtime`:

```
cd /var/cache/bind/etc
cp /etc/ld.so.cache .
cp /etc/localtime .
```

- The BIND configuration should also be placed under the new root, as `named` might need it when you reload the server. Given that your new root is set to `/var/cache/bind` you should copy the file over to `/var/cache/bind/etc/bind`.

  Note that all directories in the new `named.conf` - as specified with the `directory` option - are relative to the new root directory, so you do not need (or should) include the prefix `/var/cache/bind` there.

- The new `lib` directory must contain at least all the shared libraries used by BIND. You can list these with the **ldd** command. For instance, suppose the command:

```
ldd /usr/sbin/named
```

  yields

```
libc.so.6 => /lib/libc.so.6 (0x40015000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

  In the (real) `/lib` directory, `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 `rndc` program might be useful too. For example:

```
cp -p /usr/sbin/named{,-xfer} /var/cache/bind/usr/sbin
cp -p /usr/sbin/rndc /var/cache/bind/usr/sbin
```

Running BIND chrooted

To start the BIND nameserver in a *chroot jail*, simply add the `-t` option followed by a directory name, either on the command line, in the startup script, or in the `sysconfig` file.

Note
The `-t` option actually runs the **chroot** command to start the nameserver. Read the manual pages on the **chroot** command to learn more.

In some systems you may need to add the option to the startup file. For example:

```
start-stop-daemon ... --exec /usr/sbin/named -- -t /var/cache/bind
```

or

```
daemon ... /sbin/named -t /var/cache/bind
```

Important
The BIND nameserver switches to the chroot immediately after command line argument parsing, so before configuration files are read. All paths in the configuration file should be relative to the new root.

Configuration for a chrooted BIND

As a consequence of a chrooted jail, all configuration files (e.g. zone files and `named.conf`) must be present in a directory (e.g., `/etc/bind`) under the chroot directory.

Logging to syslog will not work either as the chrooted process has no access to the Unix socket located outside the jail (no access to `/dev/log`). There are various workarounds but the simplest is to use an alternative logfile inside the chrooted environment. See this fragment of a `named.conf` file you could use:

```
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`.

Zone files, if any, should also be put inside the chroot environment. For instance, if `named.conf` contains a definition of a master zone like this:

```
zone "example.com" IN {
    type master;
    file "/etc/bind/example.com.zone";
};
```

Then the `example.com.zone` file needs to be located in `/var/cache/bind/etc/bind`, again assuming your chroot directory is `/var/cache/bind`.

Combining special user and chroot

More technical security can be obtained by using a combination of a chrooted environment and running the nameserver under a special non-privileged user and group. Note that these users will have to be available in the copies of `/etc/passwd` and `/etc/group` that were created in the chrooted environment. Finally, set permissions on the chrooted zone and configuration files to make them read-only for the named user and group.

Note
Inspecting the source for BIND reveals the order of things:

- parsing of command line options
- chrooting
- become a background process and write pid file
- change to new user and group

The pid file is still written by the `root` user, after that the less privileged 'named' user will be used.

The `/etc/passwd`, `/etc/shadow`, and `/etc/group` files in the chroot environment may differ from the ones used by the rest of the system. The special user and group for the nameserver may not even exist outside the jail.

Securing nameserver connections

Restricting access to trusted hosts can be done using the `allow-transfer` and `allow-query` statements in `named.conf` and may help limit risks. Since the restrictions are IP address based, there is still a residual risk that a villain use spoofing techniques. This could still lead to the misuse of your system.

Signing data could help prevent this. When the server and client use a shared secret to sign the data between them we can be quite sure that both parties are whom they say they are.

To further secure nameserver connections a set of protocols called *DNSSEC* (Domain Name System Security Extensions) can be used as an extra security layer.

A cryptography crash course

To understand how DNSSEC works you need to be aware of the concepts of message authentication codes, hashes and asymmetrical encryption.

You are probably aware of the concept of symmetrical encryption. Using a key and some algorithm a cleartext message is scrambled into something incomprehensible. To restore the message, you need the same key. This works fine to shield a message from prying eyes, but it is not usable to prove that a message was sent by a given party. Either party could write a message, encrypt it with the private key and say it was the other party that wrote the message. There is no way you can tell if this is actually true.

The related concept of asymmetrical encryption also allows enciphering and deciphering of messages. But instead of one shared key there are two separate but related ones: one is used to encipher a message, the other to decipher it. After generating such a key pair the owner hands out only one of these keys to the other party or parties, the *public* key. The other half of the key pair is kept secret (the *private* key). Whomever has access to the public key can use it to encrypt a message. The owner of the corresponding private key will be the only person that can decipher it. Hence the sender can be sure that he alone can read the message. Moreover he can send an encrypted message, using the private key to encipher it. Everyone with access to the public key will be able decipher it, and can be sure that it really was the owner of the corresponding private key that sent the message.

Either private, public, or symmetric keys may be used to compute a MAC (Message Authentication Code). The algorithm to compute a MAC uses two inputs: the key and the message. The output is a fixed length string of hexadecimal digits. The MAC is computed using the key (or half of a key pair) and both the MAC and the message are subsequently sent. The receiving party will also use the message to calculate the MAC with the shared key or the other half of the key pair. If both hashes match there is a very high certainty the message has not been altered and that it originates from the sender. Note that when you use a MAC the message itself will *not* be encrypted, it can be read by anybody.

DNSSEC signs its messages with a MAC. It uses a private key that is only known to DNSSEC. The corresponding public key is published on the nameservers of the next level zone. It can be obtained by querying the parent domain nameservers (the 'trust anchor'). So the public key for example of the `sidn.nl` domain will be available on the 'nl' nameservers. The public key itself in turn will be signed using a private key whose public key will be available on the next level above. So for the domain a.b.c the

public keys of “a” can be found on the zone server for “b”. The public keys are signed with the private key whose public key is published on the nameservers of “c” (You may want to read that sentence again ;-)).

Eventually the verification chain ends at the root servers. To obtain the keys used to sign the top level domains the **dig** could be used but of course, the query might be also be spoofed. Therefore serious validating resolvers obtain the keys by other means. For instance by retrieving them from a signed software package, similarly to the built-in SSL root certificates in browsers.

Using the dnssec-keygen command

According to the lpic2 objective you must be able to use BIND 9’s **dnssec-keygen**.

Note
The **dnssec-keygen** command is part of the DNSSEC security extension for DNS (fully described in Albitz01).
DNSSEC was not fully implemented in BIND versions older than 9.2.
The 13 authoritive root servers use DNSSEC since May 6th 2010.

The **dnssec-keygen** may be used to generate public/private and symmetric keys. See CrashCryptoCourse. The key pairs are meant to be used to authenticate zone data. The symmetric keys are for Request/Transaction signatures.

The required parameters for generating a key:

a *algorithm* (`-a` option) RSA | RSAMD5 | DH | DSA | RSASHA1 | HMAC-MD5

a *key size* (`-b` option) Depends on chosen algorithm:

RSAMD5: [512..4096]
RSASHA1: [512..4096]
DH: [128..4096]
DSA: [512..1024] and a multiple of 64
HMAC-MD5: [1..512]

a *nametype* (`-n` option) ZONE | HOST | ENTITY | USER | OTHER

Format of the key files

dnssec-keygen creates two files: `K`*name*`+` *algorithm*`+` *footprint*`.private` and `K`*name*`+` *algorithm*`+` *footprint*`.key`.

As an example, say we typed this command:

```
$ dnssec-keygen -a DSA -b 768 -n ZONE example.com.
```

After completion there will be two files, which have names similar to `Kexample.com.+003+58649.key` and `Kexample.com.+003+58649.private`.

The private key file `Kkey.example.com.+003+58649.private` will have contents similar to this:

```
Private-key-format: v1.2
Algorithm: 3 (DSA)
Prime(p): ww4 ..
Subprime(q): qHwn ..
Base(g): ncWqWFJ ...
Private_value(x): YuO ...
Public_value(y): JKvGZ ...
```

Note that lines that have been truncated for better readability end in an ellipsis.

The contents of `Kkey.example.com.+003+58649.key` will be similar to this:

```
example.com. IN DNSKEY 256 3 3 BKh8J+a ...
```

Please note that generation of a HMAC-MD5 generates two files too, but both will contain the exact same 'key':

```
$ dnssec-keygen -a HMAC-MD5 -b 512 -n HOST peek.a.boo
Kpeek.a.boo.+157+39506

$ ls Kpeek.a.boo.+157+39506.*
Kpeek.a.boo.+157+39506.key  Kpeek.a.boo.+157+39506.private

$ cat Kpeek.a.boo.+157+39506.key
peek.a.boo. IN KEY 512 3 157 HUcmGO7VZ ...

$ cat Kpeek.a.boo.+157+39506.private
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: HUcmGO7VZ ...
```

Using the key

You can refer to a key by adding a `key` statement in `named.conf`, and copy and paste the key you generated in that section:

```
key key.example.com. {
    algorithm "hmac-md5";
    secret "5HUcmGO7VZ ...";
};
```

The key can be read (and possibly misused) by anybody who has access rights to the `named.conf` file. Make sure it can not be read by anybody but the nameserver user.

You can also put the key statement in a separate file and include that file using the `include` statement. In this case the same warning applies with regard to permissions.

To allow a client to use the key to connect to us we need to add a `server` clause in our configuration. Say we are on a server which has IP address `224.123.400.2` and want to allow access for a server with IP address `224.123.400.1` we may use:

```
server 224.123.400.1 {
    keys key.example.com.;
};
```

On the other server (`224.123.400.1`) we need to do the same for server `224.123.400.2`: add the `key` statement and copy in the key and allow the other host to connect to us.

You can now configure the types of access that are allowed when one has a key. As an example, to limit zone-transfers to hosts that have the proper keys installed, use:

```
zone "example.com" {
    type master;
    file "example.com.zone";
    allow-transfer { key key.example.com.; };
};
```

See the `named.conf(5)` manual page for additional use cases.

dnssec-signzone

Similar to the **dnssec-keygen** command, BIND 9 includes the **dnssec-signzone** utility. As its name implies **dnssec-signzone** can be used to sign a zone. We already saw how keys are used to *authorize* servers. Signing a zone helps a client to verify the *integrity* of the data.

When using DNSSEC, so called `RRSIG` records store the actual digital signatures that were created while signing the resource records for a domain using a `dnskey`.

The **dnssec-signzone** utility can be used to add RRSIG records to zones.

`NSEC` records (`Next SECure`) are used to specify a range of *non-existing domains*. NSEC records were invented to solve a problem: if you queried a secured nameserver (one that signs the results of a query) and you would ask for a non-existing name, it would simply return - nothing. As you can't sign 'nothing' you would never know if the anwer you got - nothing - really is what you should have gotten. Hence NSEC records. A NSEC record (Next SECure) states a range of names that do not exist. As NSEC records can be signed the answer can be trusted again.

An Example of an NSEC record for the dnssec-tools.org domain:

```
adonis.zonetransfer.me.       10800   IN      NSEC   helena.zonetransfer.me.
```

The record specifies that no DNS records exist between `adonis.zonetransfer.me.` and `helena.zonetransfer.me.`, which would exclude `brutus.zonetransfer.me.` from being an existing and therefore valid domain.

Use of NSEC records solves one problem, but introduces the next: it allows easy retrieval of all zone data by simply asking for a random name. This works because NSEC records assume alphabetical ordering of domain names. An example: if you would request 'a.example.com' and the answer would be 'do not have it, but the next valid record is joe.example.com', you know that joe.example.com exists and nothing before it does. So, then you might try jof.example.com, as 'f' is the next letter in the alphabet, etc. A solution to solve this problem is the NSEC3 record. The NSEC3 record is actually a NSEC record too but it will not return the next *name*, but *the hash of the next name*. If you ask for a.example.com you will get a record that says 'not here, and this is the hash of the next valid domain name'. There are no known methods - other than brute force - to find a name that matches that hash. If you already know the next valid domain name, you can calculate its hash and verify the answer. If you do not, the data will not help you to find out.

DANE

After having set up DNSSEC for a certain domain, it is possible to make use of DNS Authenticated Named Entities: DANE. In order to understand the advantage of DANE, we need to look at the problem DANE is trying to address. And this problem involves Certificate Authorities, or CA's in short.

The problem with CA-dependant encryption solutions lies in the implementation. When visiting a SSL/TLS Encrypted website via HTTPS, the browser software will gladly accept ANY certificate that uses a matching CN value AND is considered as being issued by a valid Certificate Authority. The CN value is dependent on DNS, and luckily we just set up DNSSEC to have some assurence regarding that. But, modern browsers come with over 1000 *trusted* root certificates. Every certificate being presented to the browser by a webserver, will be validated using these root certificates. If the presented certificate can be correlated to one of the root certificates, the science matches and the browser will not complain. And that's a problem.

In a perfect world, this system could provide some assurance. Unfortunately, every chain is only as strong as it's weakest link. Many known attack vectors exist in regards to integrity and confidentiality of HTTPS sessions. The Certificate Authorities are in a league of their own though. Just as every CA needs to be in control of their security 100% of the time, an attacker only needs one moment one CA is not. If an attacker gets hold of one of the keys used to sign certificates, it is Game Over[tm]. This has happened in the past, and there is no assurence it will not happen again in the future.

The problem consists of browsers accepting any *valid* certificate for any domain. And lack of control about exactly which certificates are valid for exactly what domain. There are a couple of workarounds out there like *Certificate Transparency* and *Certificate Pinning*. But for now, these remain to be workarounds for a flawed system. After all, the problem is not limited to your Desktop. The provided trust in CA-issued certificates becomes a problem when that CA cannot be trusted anymore. This has happened.

The infamous "Black Tulip" case describes a breach that has occured at a Dutch CA called DigiNotar. Attackers gained access to systems and generated fake certificates for major websites. Except, these certificates were not recognized as fake since they had been generated by a *trusted* CA. The *certificate chain* could be backtraced to DigiNotar and seemed valid. Despite DigiNotar never having received the order or permission to generate certificates for the affected domains. After it was discovered that attackers had issued multiple DigiNotar certificates, the trust in DigiNotar as a CA could not be maintained. The affected CA root certificate was removed from the list of trusted certificates by many software vendors. DigiNotar filed for bankcrupcy within weeks.

If that already sounds bad, consider the total impact of this data breach. Because certificates issued by DigiNotar were blacklisted as a result, software depending on valid certificates could no longer function properly. This had real-world consequences, since DigiNotar also issued certificates to government and trading related customers. The tulips could not be shipped and went black,

so to say. For this reason, some large corporations that depend on their online presence for core business get their certificates from more than one CA. This way, if one CA gets compromised and all related certificates have to be revoked, that will not be a Single Point Of Failure to the business. Good for the business, not good for transparency in regards to certificates.

This is where DANE comes in; While depending on the assurance provided by DNSSEC, DNS records are provided with *certificate associating* information. This mitigates the dependency regarding static root certificates for certain types of SSL/TLS connections. These records have come to be known as *TLSA* records. As with HTTPS, DANE should be implemented either correctly or better yet not at all. When implemented correctly, DANE will provide added integrity regarding the certificates being used for encryption. By doing so, the confidentiality aspect of the session gets a boost as well.

The TLSA *Resource Record* syntax is described in RFC 6698 sections 2 and 7. An example of a SHA-256 hashed association of a PKIX CA Certificate taken from RFC 6698 looks as follows:

```
_443._tcp.www.example.com. IN TLSA (
      0 0 1 d2abde240d7cd3ee6b4b28c54df034b9
            7983a1d16e8a410e4561cb106618e971 )
```

Each TLSA Resource Record (RR) specifies the following fields in order to create the *certificate association*: *Certificate Usage*, *Selector*, *Matching Type* and *Certificate Association Data*. The Certificate Association Data Field in the example above is represented by the SHA-256 hash string value. The contents of this Data Field are dependent on the values preceeding from the previous three RR fields. These three fields are represented by `'0 0 1'` in the example above. We will explain these fields in reverse order. This makes sense if you look at the hash value and then read the values from right to left as `'1 0 0'`.

The value `'1'` in the third field from the example above represents the *Matching Type* field. This field can have a value between `'0'` and `'2'`. It specifies whether the *Certification Association Data* field contents are NOT hashed (value `0`), hashed using SHA-256 (value `1`) or hashed using SHA-512 (value `2`). In the example above, the contents of the Certificate Association Data field represent a SHA-256 hash string.

The second field represented by a `'0'` represents the *Selector Field*. The TLSA Selectors are represented by either a `'0'` or a `'1'`. A field value of `'0'` indicates that the Certificate Association Data field contents are based on a full certificate. A value of `'1'` indicates that the contents of the Certificate Association Data Field are based on the Public Key of a certificate. In the example above, the Selector field indicates that the SHA-256 hash string from the Certificate Association Data field is based on a full certificate.

The first field represented by a '0' in the example above represents the *Certificate Usage* field. This field may hold a value between '0' and '3'. A value of `'0'` (PKIX-TA) specifies that the Certificate Association Data field value is related to a public Certificate Authority from the X.509 tree. A value of `'1'` (PKIX-EE) specifies that the Certificate Association Data field value is related to the certificate on the endpoint you are connecting to, using X.509 validation. A value of `'2'` (DANE-TA) specifies that the Certificate Association Data field value is related to a private CA from the X.509 tree. And a value of `'3'` (DANE-EE) specifies that the Certificate Association Data field value is related to the certificate on the endpoint you are connecting to. In the example above, the Certificate Usage field indicates that the certificate the SHA-256 string is based on belongs to a public Certificate Authority from the X.509 tree. Those are the same Certificate authorities that your browser uses. Field values of `'0'`, `'1'` or `'2'` still depend on these CA root certificates. The true benefit of TLSA records gets unleashed when DNSSEC is properly configured and using a value of `'3'` as a value for the Certificate Usage Field.

Looking back at the very start of the TLSA Resource Record example above, the syntax clearly follows the `_port._protocol.subject` syntax. The TLSA Resource Record always starts with an underscore '_'. Then the service port is defined (443 for HTTPS) and a transport protocol is specified. This can be either tcp, udp, or sctp according to RFC 6698. When generating TLSA records, it is also possible to specify dccp as a transport protocol. RFC 6698 does not explicitly mention dccp though. The subject field usually equals the servername and should match the CN value of the certificate.

It is not mandatory for the LPIC-2 exam to know all these details by heart. But, it is good to have an understanding about the different options and their impact. Just as using a value of `'3'` as the Certificate Usage can have some security advantages, a value of `'0'` as the Matching Type field can result in fragmented DNS replies when complete certificates end up in TLSA Resource Records. The success of security comes with usability.

To generate your own TLSA records, a page like the following can be used and should provide some insight in to the used parameters and values: www.huque.com/bin/gen_tlsa

As much as the previously mentioned website is suitable for creating occasional TLSA records, there are circumstances when custom tooling is more appropriate. The **hashslinger** command-line tool has been developed to ease the creation of TLSA records.

When implemented correctly and supported widely, DANE has the potential to secure certificate based encryption for a variety of services.

Internal DNS

If your organization uses TCP/IP on its internal networks – a very common situation nowadays – they will also need a method to locally resolve hostnames into IP addresses and vice versa. In smaller organizations whose infrastructure typically does not change much over time they might use static files (hostfiles), but given the low break-even point between the costs of maintaining such an infrastructure and the costs of installing and maintaining your own DNS most organizations run an internal DNS system.

In simpler setups, only one nameserver may suffice, though given the low costs involved nameservers will often be installed in pairs. Resolvers aware of more than one nameserver will query the other server if the primary server does not respond. If you have a number of divisions and various interconnected networks your setup will start resembling the Internet. But there are important differences.

Suppose your division is located outside the firm's main building. The workgroup will have its own nameservers. Internally, hosts will be members of the `exworks` (short for *exampleworks*) domain, that is, directly under the root (`.`) domain. To prevent accidental routing on the Internet the company chooses the use of one of the non-routable (private) IP networks: `192.168.72`. This implies we need to maintain both the domain zone and the reverse lookup zone `72.168.192.in-addr.arpa`.

We can't use the Internet infrastructure as their root nameservers don't know anything about the `exworks` and `72.168.192.in-addr.arpa` zones, nor should they: we chose to use a private network after all. If we also like to be able to resolve real Internet hostnames and addresses, we can use what is known as a *split-level* DNS setup. Two split-level DNS setups will be shown.

Limiting negotiations

We can use a feature introduced into BIND in the days of dial-up connections. BIND behind a dial-up connection can be a nuisance: every time a DNS wants to communicate with a root nameserver, an automatic dialer sets up a connection. This is expensive. You can stop BIND from doing this by putting

```
// prevent dialup access
heartbeat-interval 0;
dialup yes; // Actually disables dialup!
```

inside the `options` statement of the `named.conf` file. It is a bit counterintuitive to have to put `dialup yes` in a configuration actually meant to prevent dialups, but that's the way to do it.

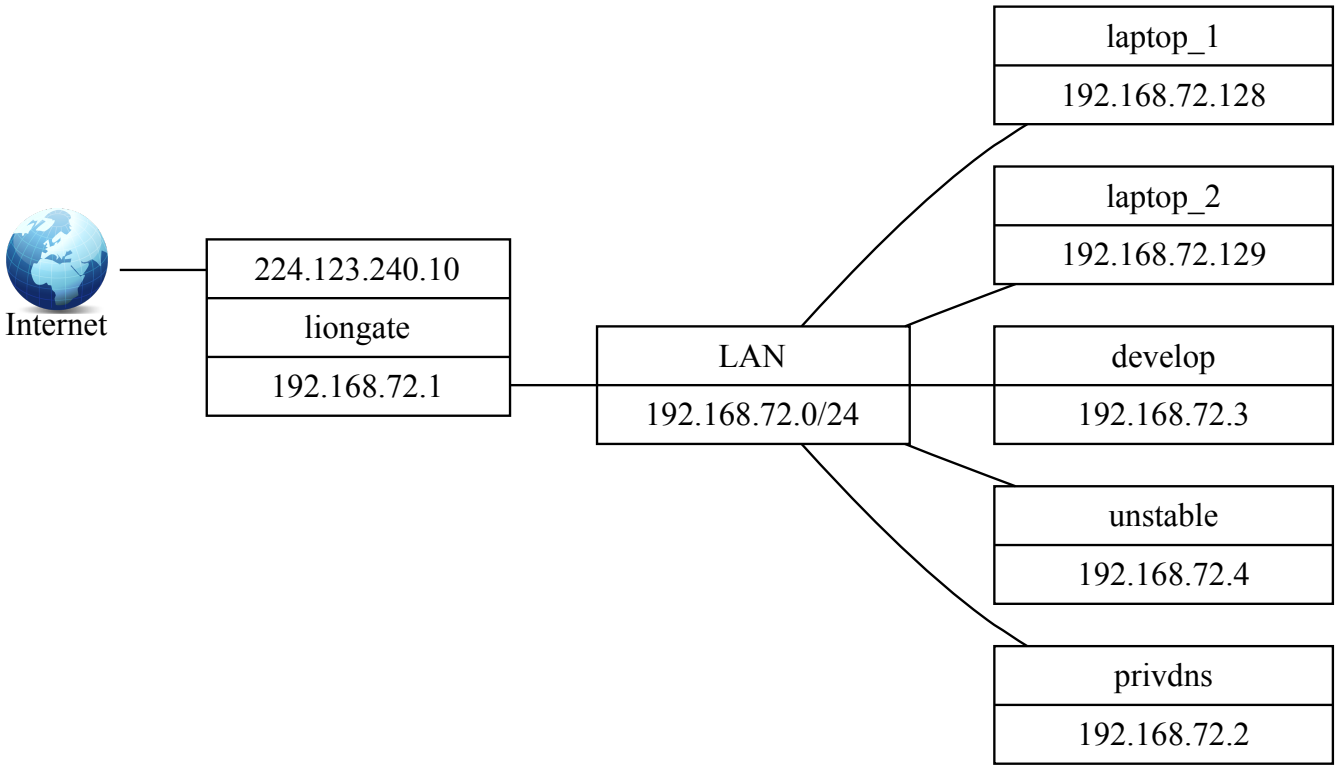
This trick can also be used to limit communication from an internal zone like `exworks`. The `exworks` and `72.168.192.in-addr.arpa` zones are implemented as conventional master zones as described earlier in this chapter. The fundamental problem with this is that the real root servers still don't delegate the `exworks` (and corresponding reverse) domain.

Therefore we should pull the internal zones together with the root nameserver definition into a separate DNS implementation. This requires at least two independent nameservers, 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 nameserver (that is, master for the internal zones) and another nameserver 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 nameservers in this example will be on different hosts. The first nameserver runs on `privdns` and will provide internal zone information. The other nameserver is on `liongate`, which will be a forwarding nameserver 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 nameserver 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 nameservers 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 as being hosted on nameserver 192.168.72.2.

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 nameservers, 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 nameserver 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 nameserver 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
```

ensures that local queries are run over the local loop interface, which is more efficient than using the outside IP address.

Note
The entries in `resolv.conf` on `privdns` should not point to the nameserver 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 nameserver 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 nameserver 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 nameservers 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 nameserver on `liongate` can be configured as `slave` for the `exworks` domain and corresponding reverse domain. We will discuss this in the next section.

Alternatives

There are two alternatives to this setup. Both require two separate nameservers.

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 nameserver entry-point.

The second alternative is to use a `slave` setup for the `exworks` (plus corresponding reverse) domain. This situation is required when two nameservers 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

We assume the same network again but with one important difference: host `privdns` is *not* available. This implies that the *internal* nameserver must now run on one of the other hosts, which also needs access to the outside world.

Both nameservers will run on the same host. At least one of the nameservers must run chrooted.

Two nameservers on one host

The following settings must be made:

- **The internal nameserver** There is a master nameserver 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 nameserver will be referred to as the *internal* nameserver.
- **The visible nameserver** The other nameserver does not need to 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 nameserver will be referred to as the *visible* nameserver, since this is the nameserver that will be visible to other hosts.

The `nameserver` line in the `resolv.conf` file for the visible nameserver 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 nameserver

The nameserver is put into a chroot jail with his own user and group ID's. The nameserver will chroot to `/var/cache/bindInternal`. It will run, for example, as user `inamed` with UID `5301` and as group with the same name and GID `5301`.

This chrooted nameserver should be able to write to some directories. To allow this, these directories should be owned by 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 nameserver should only listen on port `5353` on the internal address `127.0.0.1`. The other two, `recursion` and `fetch-glue`, prevent resolving anything except the zones that were defined here.

The root zone should does not allow queries and transfers, as it is meant for local use only.

The `exworks` and `72.168.192.in-addr.arpa` zone definitions allow zone transfers and queries originated from the other nameserver (the visible one). No other transfers and queries are allowed for these zones.

Configuring the visible nameserver

This section will describe how to configure the *visible* nameserver (if in doubt read Section 7.3.10.3.1 again). Remember it does not run chrooted, nor does it use an alternate port. It does run under its own UID/GID though.

The visible nameserver should be able to answer queries about both the inside and outside worlds.

Note
The *visible* nameserver on `liongate` could use forwarding to connect to the internal nameserver, given the software allows you to specify a port.

The `named.conf` for the visible nameserver, 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 nameserver 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:

- a nameserver to perform 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 nameserver. I.e. if the nameserver runs 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 Section 7.3.10.2.1). 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* nameserver 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 nameserver(s) will be contacted at port `53`.

A problem

A master nameserver will send a *notify* message to a slave nameserver whenever a zone definition changes. This causes the slave to initiate a zone transfer. But as both servers listen on the same IP address, the only thing to distinct them are their port numbers. Alas you can't specify a port number in a `also-notify` statement, so there is no way to notify the slave if the master data has changed.

There is a simple solution to this problem: if the master zone changes and after the internal nameserver has reloaded the zone data, restart the visible nameserver too.

Linux supports binding more than one IP address to a network interface. Another approach would be to give both servers a unique local IP address. This involves other configuration changes, which by now you should be able to manage. Therefore the exercise on how to do this is left to the reader.

TSIG

TSIG (Transaction SIGnatures) provides a secured communication channel between nameservers for zone transfers. It is a resource friendly add-on, available in BIND version 8.2 and higher.

TSIG uses shared keys and one-way hashing to provide authenticity and integrity. See CrashCryptoCourse for a short introduction to these concepts.

Configuring TSIG for BIND

We need a HMAC-MD55 key pair, which can be generated using the now familiar **dnssec-keygen** command:

```
# dnssec-keygen -a HMAC-MD5 -b 512 -n HOST rndc-key
Krndc-key.+157+24876
```

The utility will generate two output files. Note that the keys in both files will always be duplicates as we required a HMAC-MD5.

```
# ls -l Krndc-key.+157+24876*
-rw------- 1 root bind 117 Jul  5 05:28 Krndc-key.+157+24876.key
-rw------- 1 root bind 229 Jul  5 05:28 Krndc-key.+157+24876.private
```

The contents of the private keyfile might be similar to this:

```
# cat Krndc-key.+157+24876.private
Private-key-format: v1.3
Algorithm: 157 (HMAC_MD5)
Key: XIQDYlGaIbWfyopYHS1vtFrlfJiiIkiEbiNj4kN8Ke+F0hEqA7KVwcJMR/6 ↩
    URez32XBEmKZf2W1GUzjpbI2KJQ==
Bits: AAA=
Created: 20130705102726
Publish: 20130705102726
Activate: 20130705102726
```

Create a file to hold the secret you just generated and the IP addresses of slave nameservers. The name of the file can be freely chosen, in this example `tsig.key` will be used. The syntax of this file can be learned from this example:

```
key "TRANSFER" {
    algorithm hmac-md5;
    secret "XIQDYlGaIbWfyopYHS1vtFrlfJiiIkiEbiNj4kN8Ke+F0hEqA7KVwcJMR/6 ↩
        URez32XBEmKZf2W1GUzjpbI2KJQ==";
};

# nameserver 2 (slave)
server 10.0.1.2 {
    keys { TRANSFER; };
};

# nameserver 3 (slave)
server 10.0.1.3 {
    keys { TRANSFER; };
};
```

Note: the server option points to the slave nameserver.

Edit the BIND main configuration file `named.conf` and include the file like this:

```
include "/etc/bind/tsig.key";
```

Now, reload the BIND configuration:

```
# rndc reload
server reload successful
```

Use **rndc tsig-list** to list the currently active TSIG keys. This completes the configuration on the master. Next we will need to configure the slaves.

To setup the slave nameserver, create a similar `tsig.key` file there, that contains the same secret as used on the master nameserver. However, now make the `server` option point to the master nameserver:

```
key "TRANSFER" {
    algorithm hmac-sha1;
    secret "XIQDYlGaIbWfyopYHS1vtFrlfJiiIkiEbiNj4kN8Ke+F0hEqA7KVwcJMR/6 ↩
        URez32XBEmKZf2W1GUzjpbI2KJQ==";
};

# nameserver 1 (master)
server 10.0.1.1 {
    keys { TRANSFER; };
};
```

As before, include the path to the `tsig.key` file to the BIND main configuration file and reload the configuration. TSIG is now enabled.

Questions and answers

Domain Name Server

1. *What is the difference between a caching-only name server and a normal one?*

   A caching-only name server does not serve out zones, except for a few internal ones. Caching-only name server [164]

2. *Does the option **forward** in the `named.conf` file operate without further configuration?*

   No. For it to work, filling in the **forwarders** option is also required. Use a list of forwarders [166]

3. *Which syntax is used in **bind** to route various types of data to the logging output channels?*

   The **category** *type* command is used in the **logging** statement. It is followed by the output channel of choice, e.g.: **{ default_syslog; }**. The category command [167]

4. *What is @ called when used inside a zone file?*

   The @ refers to the current origin which is defined in `named.conf`. The use of @ in zone file [167]

5. *In which manner would you prompt the name server to reload its configuration and zone files?*

   The command to be used is **rndc reload**. Reloading the nameserver configuration [168]

6. *For which type of DNS servers is a special zone defined using the type **hint**?*

   For the root servers. These are listed in the root zone (**zone "."**) and statically defined as *hints* (**type hint;**). These are the starting point for looking up addresses. Hints file [176]

7. *The **SOA** record in a zone file contains - among others - a serial number. Is there a required format for this serial number?*

   Yes, but only in the sense that it is a natural number. The serial number must be incremented (by at least one) each time something is changed in order for this change in the zone to be implemented. For a zone that never changes, a single 1 is enough. However, the format **yyyymmddee** is rather common. Serial number [178]

8. *What does the **CNAME** resource record do?*

   It specifies a named alias for a host with an **A** (address) record. The CNAME record [179]

9. *Name the purpose of a reversed zone when used by the **host** command.*

   To map an IP address to the corresponding hostname. Reverse zone files [180]

10. *How is a zone defined as **master**?*

    A zone is defined as master by using the **type master** statement within a zone definition. Define a zone as master [182]

11. *What two other DNS security strategies can be applied besides the obvious* security by obscurity *measures?*

    One is limiting the effects of an intrusion by running BIND with *less privileges* or by running it in a *chroot environment* or *jail*. The other is securing name server connections by *signing reponses* sent by the name server. Running BIND with less privileges or in a chroot jail [192]

12. *Name at least two categories which are distinguished by the **category** command in the **logging** statement.*

    `security`, `lame-servers`, `cname` are all valid categories. Some logging categories [167]

13. *What could be wrong if the **dig** command, when used to test the reverse entry for a hostname, appends the current origin to the name?*

    The zone file might have an error, where there's no trailing dot appended to the hostname in the PTR record definition. Assure Trailing dot in PTR [185]

Chapter 8

HTTP Services (208)

This topic has a weight of 11 points and contains the following objectives:

Objective 208.1; Basic Apache Configuration (4 points) Candidates should be able to install and configure a web server. This objective includes monitoring the server's load and performance, restricting client user access, configuring support for scripting languages as modules and setting up client user authentication. Also included is configuring server options to restrict usage of resources. Candidates should be able to configure a web server to use virtual hosts and customize file access.

Objective 208.2; Apache configuration for HTTPS (3 points) Candidates should be able to configure a web server to provide HTTPS.

Objective 208.3; Implementing Squid as a caching proxy (2 points) Candidates should be able to install and configure a proxy server, including access policies, authentication and resource usage.

Objective 208.4; Implementing Nginx as a web server and a reverse proxy (2 points) Candidates should be able to install and configure a reverse proxy server, Nginx. Basic configuration of Nginx as a HTTP server is included.

Basic Apache Configuration (208.1)

Candidates should be able to install and configure a web server. This objective includes monitoring the server's load and performance, restricting client user access, configuring support for scripting languages as modules and setting up client user authentication. Also included is configuring server options to restrict usage of resources. Candidates should be able to configure a web server to use virtual hosts and customise file access.

Key Knowledge Areas

Apache 2.x including 2.4 configuration files, terms and utilities

Apache log files configuration and content

Access restriction methods and files

mod_perl and PHP configuration

Client user authentication modules, files and utilities

Configuration of maximum requests, minimum and maximim servers and clients

Apache 2.x virtual host implementation (with and without dedicated IP addresses)

Using redirect statements in Apache's configuration files to customise file access

Terms and utilities

- `access.log` or `access_log`
- `error.log` or `error_log`
- `.htaccess`
- `httpd.conf`
- `mod_auth`
- `mod_authn_file`
- `mod_access_compat`
- **htpasswd**
- `AuthUserFile, AuthGroupFile`
- **apache2ctl**
- **httpd**

Resources: LinuxRef06; LPIC2sybex2nd; Coar00; Poet99; Wilson00; Engelschall00; PerlRef01; Krause01; apachedoc; apache24upgrad digochanges; apache24upgrade; apache24upgrade; the **man** pages for the various commands.

Installing the Apache web-server

Building Apache from source was routinely done when Apache emerged. Nowadays Apache is available in binary format for most modern (post 2005) Linux distributions. Installing programs from source is already covered in 206.1. Therefore, we will concentrate on working with rpm and apt package managers and tools during this chapter. Do not underestimate the importance of building Apache from source though. Depending on requirements and (lack of) availability, it might still be necessary to compile Apache and Apache modules from source. The Apache binary **httpd** can be invoked with certain command-line options that affect the behaviour of the server. But in general Apache is started by other scripts that serve as a wrapper for **httpd**. These scripts should take care of passing required flags to **httpd**. The behaviour of the server is configured by setting various options called `directives`. These `directives` are declared in configuration files. The location of configuration files and how they are organized varies. Red Hat and similar distributions have their configuration files in the `/etc/httpd/conf` directory. Other locations which are or have been used are `/etc/apache/config`, `/etc/httpd/config` and `/etc/apache2`.

Depending on your Linux distribution and enabled repositories, your distribution may come with Apache 2.2 or Apache 2.4 or both. Apache 2.0 does comply to the LPIC-2 Apache 2.x scope due to its name, but Apache 2.0 is no longer maintained. It is therefore not recommended to use Apache 2.0. Instead, Apache 2.4 is recommended to be used by the Apache foundation. As a Linux administrator, you may however still encounter Apache 2.0 on servers. It is therefore recommended to be familiar with the configuration differences between the different versions. The Apache foundation does provide guidence: Via https://httpd.apache.org/docs/ upgrade documents can be accessed that address the necessary steps when upgrading from Apache 2.0 to 2.2, and from Apache 2.2 to 2.4.

It is important to distinguish between (global) directives that affect the Apache server processes, and options that affect a specific component of the Apache server, i.e. an option that only affects a specific website. The way the configuration files are layed out can often be a clue as to where which settings are configured. Despite this presumed obviousness, it is also important not to make assumptions. Always familiarize yourself with all the configured options. When in doubt about a specific option, use the documentation or a web search to find out more and consider whether the option is configured appropriately.

On many (Red Hat based) distributions the main Apache configuration file is `httpd.conf`, other (Debian based) distributions favour the `apache2.conf` filename. Depending on your distribution and installation it might be one big file or a small generic one with references to other configuration files via `Include` and/or `IncludeOptional` statements. The difference between these two directives lies in the optional part. If Apache is configured to `Include` all `*.conf` files from a certain directory, there has to be at least one file that matches that pattern to include. Otherwise, the Apache server will fail to start. As an alternative, the `IncludeOptional` directive can be used to include configuration files *if* they are present and accessible. The Apache main configuration file can configure generic settings like servername, listening port(s) and which IP addresses these ports should be

bound to. There may also be a seperate `ports.conf` configuration file though, so always follow the `Include` directives and familiarize yourself with the contents of *all* configuration files. The user and group Apache should run as can also be configured from the main configuration file. These accounts can be set to switch after startup. This way, the Apache software can be started as the root user, but then switch to a dedicated "www", "httpd" or "apache" user to adhere to the principle of least privilege. There are also various directives to influence the way Apache serves files from its document tree. For example there are `Directory` directives that control whether it is allowed to execute PHP files located in them. The default configuration file is meant to be self explanatory and contains a lot of valuable information. In regards to the LPIC-2 exam, you are required to be familiar with the most common Apache directives. We shall cover some of those in the section to come. At the time of this writing, Apache 2.4 is the latest stable version and the recommended version to use according to it's distributor, the Apache Foundation. Where applicable, this book will try to point out the differences between the various versions.

An additional method to set options for a subdivision of the document tree is by means of an `.htaccess` file. For security reasons you will also need to enable the use of `.htaccess` files in the main configuration file by setting the `AllowOverride` directive for that `Directory` context. All options in an `.htaccess` file influence files in the directory and the ones below it, unless they are overridden by another `.htaccess` file or directives in the main configuration file.

Modularity

Apache has a modular source code architecture. You can custom build a server with only modules you really want. Many modules are available on the Internet and you could also write your own.

Modules are compiled objects written in C. If you have questions about the development of Apache modules, join the Apache-modules mailing list at *http://httpd.apache.org/lists.html*. Remember to do your homework first: research past messages and check all the documentation on the Apache site before posting questions.

Special modules exist for the use of interpreted languages like Perl and Tcl. They allow Apache to run interpreted scripts natively without having to reload an interpreter every time a script runs (e.g. `mod_perl` and `mod_tcl`). These modules include an API to allow for modules written in an interpreted (scripted) language.

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 during the *build* phase. Current implementations of Apache are capable of *run-time* module loading. The section on DSO has more details.

Run-time loading of modules (DSO)

Most modern Unix derivatives have a mechanism for the on demand linking and loading of so called Dynamic Shared Objects (DSO). This is a way to load a special program into the address space of an executable at run-time. This can usually be done in two ways: either automatically by a system program called `ld.so` when the executable is started, or manually from within the executing program with the system calls **dlopen()** and **dlsym()**.

In the latter method the DSO's are usually called shared objects or DSO files and can be named with an arbitrary extension. By convention the extension `.so` is used. These files are usually installed in a program-specific directory. The executable program manually loads the DSO at run-time into its address space via **dlopen()**.

Tip
How to run Apache-SSL as a shareable (DSO) module: Install the appropriate package:

```
packagemanager installcommand modulename
```

Depending on your distribution, the configuration file(s) might or might not have been adjusted accordingly. Always check for the existence of a LoadModule line in one of the configuration files:

```
LoadModule apache_ssl_module modules/libssl.so
```

This line might belong in the Apache main configuration file, or one of the included configuration files. A construction that has been receiving much support lately, is the use of seperate `modules-available` and `modules-enabled` directories. These directories are subdirectories inside the Apache configuration directory. Modules are installed in the `modules-available` directory, and an `Include` reference is made to a symbolic link inside the `modules-enabled` directory. This symbolic link then points back to the module. The `Include` reference might be a wildcard, including all files from a certain directory.
Another construction is similar, but includes a `conf.modules.d` directory inside the Apache configuration directory. This file is in fact a symbolic link, pointing to a directory inside the Apache program directory somewhere else on the filesystem. An example from a Red Hat based host:

```
Include conf.modules.d/*.conf
```

Again, the implementations you could encounter might differ significantly from each other. Various aspects such as Linux distribution used, Apache version installed or whether Apache is installed from packages or source may be of influence to the way Apache is implemented. Not to mention the administrator on duty. Important to remember is that Apache often uses configuration files that may be nested. But that there will always be one main Apache configuration file, at the top of the hiearchy.

Tip
To see whether your version of Apache supports DSOs, 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 make use of dyamic modules.

APache eXtenSion (APXS) support tool

The APXS is a new support tool from Apache 1.3 and onwards 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.

Monitoring Apache load and performance

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 is averaged.

Enhancing Apache performance

Lack of available RAM may result in memory swapping. A swapping webserver will perform badly, especially if the disk subsystem is not up to par. Causing users to hit stop and reload, further increasing the load. You can use the `MaxClients` setting to limit the amount of children your server may spawn hence reducing memory footprint. It is advised to **grep** through the Apache main configuration file for all directives that start with `Min` or `Max`. These settings define the MINimal and MAXimum boundaries for each affected setting. The default values should provide a concense balance between server load at idle on one hand, and the possibility to handle heavy load on the other. As each chain is only as strong as it's weakest link, the underlying system should be adequatetly configured to handle the expected load. The LPIC-2 exam focuses more on the detection of these performance bottlenecks in chapter 200.1.

Apache `access_log` file

The `access_log` contains a generic overview of page requests for 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 like the following:

```
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 format as shown is known as the Common Log Format (CLF). It is a standard format produced by many web servers and can be read by most log analysis programs. Log file entries produced in CLF will look similar to this line:

```
127.0.0.1 - bob [10/Oct/2000:13:55:36 -0100] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

CLF contains 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).

Apache `error_log` file

The server error log, whose name and location is set by the *ErrorLog* directive, is a very important log file. This is the file to which Apache httpd will send diagnostic information and record any errors that it encounters in processing requests. It is a good place to look when a problem occurs starting the server or while operating the server. It will often contain details of what went wrong and how to fix it.

The error log is usually written to a file (typically error_log on Unix systems and error.log on Windows). On Unix systems it is also possible to have the server send errors to syslog or pipe them to a program.

The format of the error log is relatively free-form and descriptive. But there is certain information that is contained in most error log entries. For example, here is a typical message:

```
[Wed Oct 11 14:32:52 2000] [error] [client 127.0.0.1] client denied by server \
configuration: /export/home/live/ap/htdocs/test
```

The first item in the log entry is the date and time of the message. The second item lists the severity of the error being reported. The `LogLevel` directive is used to control the types of errors that are sent to the error log by restricting the severity level. The third item gives the IP address of the client that generated the error. Beyond that is the message itself, which in this case indicates that the server has been configured to deny the client access. The server reports the file-system path (as opposed to the web path) of the requested document.

A very wide variety of different messages can appear in the error log. Most look similar to the example above. The error log will also contain debugging output from CGI scripts. Any information written to stderr by a CGI script will be copied directly to the error log.

It is not possible to customize the error log by adding or removing information. However, error log entries dealing with particular requests have corresponding entries in the access log. For example, the above example entry corresponds to an access log entry with status code 403. Since it is possible to customize the access log, you can obtain more information about error conditions using that log file.

During testing, it is often useful to continuously monitor the error log for any problems. On Unix systems, you can accomplish this using:

```
tail -f error_log
```

Knowing how to customize Apache logging may prove to be a very usable skill. Manually reviewing Apache logs is not for the faint of heart. For a low-traffic server, this may still be doable. Otherwise, looking for information by sifting through logs on a busy server that serves multiple websites, can become a very intense textfile-manipulating-excercise. This creates a paradox: With little to no logging, hardly any input is available when looking for the cause of a problem. With very elaborate logging in place, the information may be overwhelming. For this reason, Apache logs are often interpreted by external facilities. The logs are either sent to or read by a system that has the capability to visualize statistics and recognize patterns. To ensure the provided logging is adequate, customizing the Apache logging first may be necessary.

Apache 2.3.6 and later provide the possibility to enable different kinds of `LogLevel` configurations on a per-module or per-directory basis. The Apache documentation regarding the `Loglevel` directive is outstanding and there is not much we could add to that.

Restricting client user access

Many systems use either DAC or MAC to control access to objects:

Discretionary Access Control (DAC) A system that employs DAC allows users to set object permissions themselves. They can change these at their discretion.

Mandatory Access Controls (MAC) A system that employs MAC has all its objects (e.g., files) under strict control of a system administrator. Users are not allowed to set any permissions themselves.

Apache takes a liberal stance and defines discretionary controls to be controls based on usernames and passwords, and mandatory controls to be based on static or quasi-static data like the IP address of the requesting client.

Apache uses modules to authenticate and authorise users. First of all, the difference between authentication and authorization should be clear. Authentication is the process in which a user should validate their identity. This is the *who* part. Authorization is the process of deciding *who* is allowed to do *what*. Authorization either allows or denies requests made to the Apache server. Authorization depends on authentication to make these decisions.

The Apache modules that serve the purpose of autheNtication, follow the naming convention of `mod_authn_*`. The modules that serve the purpose of authoriZation, follow the convention of `mod_authz_*`. An exception to this rule is the `mod_authnz_ldap` module. As you might have guessed, due to the nature of LDAP this module can aid in both authentication as well as authorization.

The location of these modules on the filesystem may vary. Most distributions create a `modules`, `modules.d` or `modules-availab` directory within the Apache configuration directory. This directory can very well be a symbolic link to a directory somewhere else on the filesystem. This can be determined by invoking **pwd -P** or **ls -ld** from within the modules directory as shown by the following example:

```
[user@redhatbased /etc/httpd]$ pwd -P
/usr/lib64/httpd/modules
```

In the example above, the symbolic link `/etc/httpd/modules` provides for easy reference to the modules from within Apache configuration files. Apache modules are loaded using the `LoadModule` directive. This directive expects the path to the module to be relative to the Apache configuration directory declared by the `ServerRoot` directive.

In general, modules will use some form of database to store and retrieve credential data. The **mod_authn_file** module for instance uses text files where **mod_auth_dbm** employs a Unix DBM database.

Below is a list of some modules that are included as part of the standard Apache distribution.

mod_auth_file (DAC) This is the basis for most Apache security modules; it uses ordinary text files for the authentication database.

mod_access (MAC) This used to be the only module in the standard Apache distribution which applies what Apache defines as mandatory controls. It used to allow you to list hosts, domains, and/or IP addresses or networks that were permitted or denied access to documents. As of Apache 2.4, this module is no longer used. Apache 2.4 and newer use an updated authentication and authorization model. This new model also comes with new modules, new directives and new syntax. The `mod_access` module is still an LPIC-2 exam objective, so the pre-2.4 syntax should still be familiar to you. In order to aid the migration towards Apache 2.4, a module called `mod_access_compat` ships with Apache 2.4. This module serves the purpose of still accepting the pre-2.4 syntax on Apache 2.4 servers. If you encounter `mod_access` related errors after upgrading to Apache 2.4 from a previous version, make sure the Apache 2.4 configuration loads this compabibility module with a line similar to:

```
LoadModule mod_access_compat modules/mod_access_compat.so
```

mod_authn_anon (DAC) 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_authn_dbm (DAC) Like **mod_auth_db**, except that credentials are stored in a Unix DBM file.

mod_auth_digest (DAC) This module implements HTTP Digest Authentication (RFC2617), which used to provide a more secure alternative to the `mod_auth_basic` functionality. The explanation that follows is nice to know but outdated. The whole point of digest authentication was to prevent user credentials to travel via unencrypted HTTP over the wire. The hashing algorithms used by the digest module are however seriously outdated. Using digest authentication instead of basic HTTP authentication does not offer as many advantages in terms of security as the use of HTTPS would. The following documentation page provides more detail: http://httpd.apache.org/docs/2.4/mod/mod_auth_digest.html.

After receiving a request and a user name, the server will challenge the client by sending a `nonce`. The contents of a nonce can be any (preferably base 64 encoded) string, and the server may use the nonce to prevent replay attacks. A nonce might, for example, be constructed using an encrypted timestamp within a resolution of a minute, i.e. '201611291619'. The timestamp (and maybe other static data identifying the requested URI) might be encrypted using a private key known only to the server.

Upon receival of the nonce the client calculates a hash (by default a MD5 checksum) of the received nonce, the username, the password, the HTTP method, and the requested URI and sends the result back to the server. The server will gather the same data from session data and password data retrieved from a local digest database. To reconstruct the nonce the server will try twice: the first try will use the current clocktime, the second try (if necessary) will use the current clocktime minus one minute. One of the tries should give the exact same hash the client calculated. If so, access to the page will be granted. This restricts validity of the challenge to one minute and prevents replay attacks.

Please note that the contents of the nonce can be chosen by the server at will. The example provided is one of many possibilities. Like with **mod_auth**, the credentials are stored in a text file (the digest database). Digest database files are managed with the **htdigest** tool. Please refer to the module documentation for more details.

mod_authz_host The `mod_authz_host` module may be used to `Require` a certain source of request towards Apache. The `mod_authz_host` module is quite flexible about the arguments provided. Due to the name of the module, it may seem logical to provide a hostname. While this certainly works, it may not be the preferred choice. Not only does this module need to perform a forward DNS lookup on the provided hostname to resolve it to a numerical IP, the module is also configured to perform a reverse DNS lookup on the resolved numerical IP after the forward lookup is performed. Providing a hostname thus leads to at least two DNS lookups for every affected webserver request. And if the reverse DNS result differs from the provided hostname, the request will be denied despite what the configuration may allow. To circumvent this requirement regarding forward and reverse DNS records matching, the `forward-dns` option may be used when providing a hostname. Luckily, `mod_authz_host` not only accepts hostnames as an argument. It can also handle (partial) IP addresses, both IPv4 and IPv6, and CIDR style notations. There is also an argument available called `local`. This will translate to the `127.0.0.0/8` or `::1` loopback addresses as well as the configured IP addresses of the server. This setting may come in handy when restricting connections in regards to the local host. Because of the liberal way that IP addresses are interpreted, it is recommended to be as explicit as possible when using this module. For instance, all of the following is regarded as valid input and will be interpreted by the rules that apply:

```
Require host: snow.nl
Require ip: 10.6.6
Require ip: 172
```

```
Require ip: 10.9.9.9/32
Require forward-dns: cloudhost.snow.nl
Require local
```

One of the noteworthy differences between Apache 2.2 and 2.4 lies in the directives used for authorization. The authorization functionality is provided by Apache `mod_authz_*` modules. Where previous versions of Apache used the `Order`, `Allow from` , `Deny from` and `Satisfy` directives, Apache 2.4 uses new directives called `all`, `env`, `host` and `ip`. These new directives have a significant impact on the syntax of configuration files. In order to aid the transition towards Apache 2.4, the `mod_access_compat` module can still interpret the previously used authorization directives. This module has to be explicitly enabled though. In doing so, backwards compatibility towards previous authorization configuration directives is maintained. The current authorization directives provide the possibility of a more granular configuration in regards to who is authorized to do what. This added granularity mostly comes from the availability of the `Require` directive. This directive could already be used before Apache 2.4 for authentication purposes. Since Apache 2.4 though, this directive can also be interpreted by the authorization modules.

The following example puts the old en new syntax in comparison, while providing the same functionality.

First, the pre-2.4 style:

```
<Directory /lpic2bookdev>
Order deny,allow
Deny from all
allow from 10.6.6.0/24
Require group employees
Satisfy any
</Directory>
```

And now the same codeblock, but using the Apache 2.4 style syntax:

```
<Directory /lpic2bookdev>
<RequireAny>
Require ip 10.6.6.0/24
Require group employees
</RequireAny>
</Directory>
```

The benefit of the new syntax is all about efficiency. By accomplishing the same functionality with fewer lines, the processing of those lines will be handled more effectively by both humans and computers. The computers benefit from spending less processing cycles while accomplishing the same result. Humans benefit from a short configuration section. Long configurations are more prone to contain errors that may be overlooked. By creating sections within configuration files using the `RequireAll`, `RequireAny`, and `RequireNone` directives, these configurations can contain granular rules while at the same time preserving their readability.

Another 2.4 change that is worth mentioning, has to do with the LPIC-2 exam objective regarding the `mod_auth` module. Starting with Apache 2.1, the functionality of the `mod_auth` module has been superseeded by more specific modules. One of these modules, `mod_authn_file` now provides the functionality that was previously offered by `mod_auth`. `mod_authn_file` allows for the use of a file that holds usernames and password as part of the authorization process. This file can be created and the contents may be maintained by the **htpasswd** utility. When using `mod_auth_digest` instead of `mod_auth_basic`, the **htdigest** utility should be used instead. This book will focus on the `mod_auth_basic` option. The **htpasswd -c** option will create a file with the provided argument as a filename during creation of a username and password pair. **htpasswd** allows for the creation of crypt, MD5 or SHA1 password algorithms. As of Apache 2.4.4, it is also possible to use bcrypt as the password encryption algorithm. Plaintext passwords can also be generated using the **htpasswd -p** option, but will only work if Apache 2.4 is hosted on Netware and MS Windows platforms. The crypt algorithm used to be the **htpasswd** default algorithm up to Apache version 2.2.17, but is considered insecure. Crypt will limit the provided password to the first eight characters. Every part of the password string from the ninth character on will be neglected. Crypt password strings are subject to fast brute force cracking and therefore pose a considerable security risk. The use of the crypt algorithm should be avoided whenever possible. Instead, the bcrypt algorithm should be considered when available. On a system with Apache 2.4.4 or later, the following syntax can be used to create a new password file `htpasswdfile`, supply it with the user "bob" and set the password for the user account using the bcrypt algorithm:

```
htpasswd -cB /path/outside/document/root/htpasswdfile bob
```

The system will ask for the new password twice. To update this file anytime later by adding the user "alice", the `-c` option can be ommited to prevent the file from being rewritten:

```
htpasswd -B /path/outside/document/root/htpasswdfile alice
```

Using the brypt algorithm with **htpasswd** also enables the use of the `-C` option. Using this option, the computing time used to calculate the password hash may be influenced. By default, the system uses a setting of 5. A value between 4 and 31 may be provided. Depending on the available resources, a value up to 18 should be acceptable to generate whilst increasing security. To add the user eve to the existing `htpasswdfile` while increasing the computing time to a value of 18, the following syntax may be used:

```
htpasswd -B -C18 /path/outside/document/root/htpasswdfile eve
```

In the examples above, it is suggested that the password file is created outside of the webserver document tree. Otherwise, it could be possible for clients to download the password file.

To use the generated password file for authentication purposes, Apache has to be aware of the `htpasswdfile` file. This can be accomplished by defining the `AuthUserFile` directive. This directive may be defined in either the Apache configuration files, or in a seperate `.htaccess` file. That `.htaccess` file should be located inside the directory of the document root it should represent. The Apache config responsible for that document root should have the `AllowOverride` directive specified. This way Apache will override directives from its configuration for directories that have `.htaccess` documents in them. The syntax for the `.htaccess` documents is the same as for Apache configuration files. A code block to use for user authentication could look as follows:

```
<Directory /web/document/root>
AuthName "Authentication Required"
AuthType Basic
AuthUserFile /path/outside/document/root/htpasswdfile
Require valid-user
Documentroot /web/document/root
</Directory>
```

Consult the contents of your Apache modules directory for the presence of mod_auth* files. There are multiple authentication and authorization modules available. Each has its own purpose, and some depend on each other. Each module adds functionality within Apache. This functionality can be addressed by using specific module-specific directives. Refer to the Apache documentation website https://httpd.apache.org/docs/2.4/mod/ for detailed usage options regarding the modules available for Apache 2.4.

Configuring authentication modules

Apache security modules are configured by configuration directives. These are read from either the centralized configuration files (mostly found under or in the `/etc/` directory) or from decentralized `.htaccess` files. The latter are mostly used to restrict access to directories and are placed in the top level directory of the tree they help to protect. For example, authentication modules will read the location of their databases using the **AuthUserFile** or **AuthDBMGroupFile** directives.

Centralized configuration This is an example of a configuration as it might occur in a centralized configuration file:

```
<Directory /home/johnson/public_html>
<Files foo.bar>
AuthName "Foo for Thought"
AuthType Basic
AuthUserFile /home/johnson/foo.htpasswd
Require valid-user
</Files>
</Directory>
```

The resource being protected is "any file named foo.bar" in the `/home/johnson/public_html` directory *or any underlying subdirectory*. Likewise, the file specifies whom are authorized to access `foo.bar`: any user that has credentials in the `/home/johnson/foo.htpasswd` file.

Decentralized configuration The alternate approach is to place a `.htaccess` file in the top level directory of any document tree that needs access protection. Note that you must set the directive `AllowOverride` in the *central* configuration to enable this.

The first section of `.htaccess` determines which authentication type should be used. It can contain the name of the password or *group file* to be used, e.g.:

```
AuthUserFile {path to passwd file}
AuthGroupFile {path to group file}
AuthName {title for dialog box}
AuthType Basic
```

The second section of `.htaccess` ensures that only user `{username}` can access (GET) the current directory:

```
<Limit GET>
require user {username}
</Limit>
```

The `Limit` section can contain other directives to restrict access to certain IP addresses or to a group of users.

The following would permit any client on the local network (IP addresses 10.*.*.*) to access the `foo.html` page and require a username and password for anyone else:

```
<Files foo.html>
Order Deny,Allow
Deny from All
Allow from 10.0.0.0/8
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 files that contain lists of valid users. The **htpasswd** command can be used to create and update these files. The resulting files are plain text files, which can be read by any editor. They contain entries of the form "username:password", where the password is encrypted. Additional fields are allowed, but ignored by the software.

htpasswd encrypts passwords using either a version of MD5 modified for Apache or the older `crypt()` routine. You can mix and match.

```
SYNOPSIS
htpasswd [ -c ] passwdfile username
```

Here are two examples of using **htpasswd** for creating an Apache password file. The first is for creating a new password file while adding a user, the second is for changing the password for an existing user.

```
$ htpasswd -c /home/joe/public/.htpasswd joe
$ htpasswd /home/joe/public/.htpasswd stephan
```

Note
Using the `-c` option, the specified password file will be overwritten if it already exists!

Group files

Apache can work with *group files*. Group files contain group names followed by the names of the people in the group. By authorizing a group, all users in that group have access. Group files are known as `.htgroup` files and by convention bear that name - though you can use any name you want. Group files can be located anywhere in the directory tree but are normally placed in the toplevel directory of the tree they help to protect. To allow the use of group files you will need to include some directives in the Apache main configuration file. This will normally be inside the proper `Directory` definition. Where the `AuthUserFile` may specify either an absolute or relative path, the `AuthGroupFile` directive will always treat the provided argument as relative to the `ServerRoot`. The `AuthGroupFile` file functions as an addition to the `AuthUserFile`. The file should contain a group on each line, followed by a colon. An example:

Apache main configuration file:

```
...
AuthType Basic
AuthUserFile /var/www/.htpasswd
AuthGroupFile /var/www/.htgroup
Require group Management
...
```

The associated `.htgroup` file might have the following syntax:

```
Management: bob alice
Accounting: joe
```

Now the accounts “bob” and “alice” would have access to the resource but account “joe” would not due to the “Require group Management” statement in the main configuration file because “joe” is not a member of the required “Management” group. For this to work the users specified in the `.htgroup` file must have an entry in the `.htpasswd` file as well.

Note
A username can be in more than one group entry. This simply means that the user is a member of both groups.

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.

Note
Make sure the various files are readable by the webserver.

Configuring mod_perl

mod_perl is another module for Apache, which loads the Perl interpreter into your Apache webserver, reducing spawning of child processes and hence memory footprint and need for processor power. Another benefit is code-caching: modules and scripts are loaded and compiled only once, and will be served from the cache for the rest of the webserver’s life.

Using **mod_perl** allows inclusion of Perl statements into your webpages, which will be executed dynamically if the page is requested. A very basic page might look like this:

```
print "Content-type: text/plain\r\n\r\n";
print "Hello, you perly thing!\n";
```

mod_perl also allows you to write new modules in Perl. 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. The first is `Apache::`

`Registry`, which can transparently run well-written existing perl CGI scripts. If you have badly written scripts, you should rewrite them. If you lack resources, you may choose to use the second module `Apache::PerlRun` instead because it doesn't use caching and is far more permissive then `Apache::Registry`.

You can configure your **httpd** server and handlers in Perl using `PerlSetVar`, and `<Perl>` sections. You can also define your own configuration directives, to be read by your own modules.

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 Guide PerlRef01.

Building Apache from source code

For building Apache from source code you should have downloaded the Apache source code, the source code for **mod_perl** and have unpacked these in the same directory [1]. 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 following 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 `httpd.conf` 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 starts. During startup, 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.example.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 instead of displaying this file. Any URI of the form `http://www.example.com/perl/file.pl` will now 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. Setting `PerlSendHeader` to on tells the server to send an HTTP headers to the browser on every script invocation but most of the time it's better either to use the `$r->send_http_header` method using the Apache Perl API or to use the `$q->header` method from the `CGI.pm` module.

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. Later he rewrote his code in C and hence the "Personal Home Page/Forms Interpreter" (PHP/FI) was born.

[1] The **mod_perl** module can be obtained at perl.apache.org, the source code for Apache at www.apache.org

Over the next two to three years, it evolved into PHP/FI 2.0. Zeev Suraski and Andi Gutmans wrote a new parser in the summer of 1997, which led to the introduction of PHP 3.0. PHP 3.0 defined the syntax and semantics used in both versions 3 and 4. PHP became the de facto programming language for millions of web developers. Still another version of the (Zend) parser and much better support for object oriented programming led to the introduction of version 5.0 in july 2004. Several subversions followed and also version 6 was started to include native Unicode support. However this version was abandoned. For the year 2015 the start for version 7.0 was planned.

PHP can be called from the CGI interface, but the common approach is to configure PHP in the Apache web server as a (dynamic) DSO module. To do this, you can either use pre-built modules extracted from RPM's or roll your own from the source code[2]. 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 may want to search for these lines and uncomment them:

```
Addtype application/x-httpd-php .php
```

Then restart Apache by issuing the **apachectl restart** command. The **apachectl** command is another way of passing commands to the Apache server instead of using **/etc/init.d/httpd**. Consult the `apachectl(8)` manpage for more information.

To test whether it actually works, create the following page:

```
<HTML>
<HEAD><TITLE>PHP Test </TITLE></HEAD>
<BODY>
<?php phpinfo( ) ?>
</BODY>
</HTML>
```

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. Notice that PHP commands are contained by `<?` and `?>` tags.

The httpd binary

The **httpd** binary is the actual HTTP server component of Apache. During normal operation, it is recommended to use the **apachectl** or **apache2ctl** command to control the httpd daemon. On some distributions the **httpd** binary is named **apache2**.

Apache used to be a daemon that forked child-processes only when needed. To allow better response times, nowadays Apache can also be run in pre-forked mode. This means that the server will spawn a number of child-processes in advance, ready to serve any communication requests. On most distributions the pre-forked mode is run by default.

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.

[2] The source code for PHP4 can be obtained at www.php.net

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 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`, the superfluous spares are killed.

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.

Note
In most Red Hat derivates the Apache configuration is split into two subdirectories. The main configuration file `httpd.conf` is located in `/etc/httpd/conf`. The configuration of Apache modules is located in `/etc/httpd/conf.d`. Files in that directories with the suffix `.conf` are added to the Apache configuration during startup of Apache.

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 HTTP server relies on the client (e.g. the browser) to report the hostname as part of the HTTP request headers. By using name-based virtual hosting, one IP address may serve multiple websites for different web domains. In other words: Name-based virtual hosts use the website address from the URL to determine the correct virtual host to serve.

* **IP-based virtual hosting** Using IP-based virtual hosting, each configured web domain is committed to at least one IP address. Since most host systems can be configured with multiple IP addresses, one host can serve multiple web domains. Each web domain is configured to use a specific IP address or range of IP addresses. In other words: IP-based virtual hosts use the IP address of the TCP 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 domain name to the correct IP address first. Then, configure the Apache HTTP Server to recognize the different domain names and serve the appropriate websites.

Tip
Name-based virtual hosting eases the demand for scarce IPv4 addresses. Therefore you could (or 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. On Apache 2.x up to 2.4, this is configured using the `NameVirtualHost` directive. This NameVirtualHost directive is deprecated since Apache 2.4. Each `VirtualHost` also implies a NameVirtualHost, so defining a VirtualHost is sufficient from Apache 2.4 on. Any available IP address can be used. There should be a balance between ease of configuration, use and administration on one hand, and security on the other. Using a wildcard as the listening IP address inside a `NameVirtualHost` or `VirtualHost` segment will enable the functionality of that specific configuration on all IP addresses specified by the `Listen` directive of Apache's main configuration file. If the main configuration file also uses a wildcard for the `Listen` option, this will result in the availability of the Apache HTTPD server on all configured IP addresses of the server.

And therefore, the availability of the previously mentioned functionality on all of these IP addresses as well. Whether or not this is either preferable or imposes risk, depends on the circumstances. If the server is using multiple network interfaces and/or IP addresses, special care should be taken when configuring services. Every daemon exposing services to the network could contain code based or configuration errors. These errors could be abused by someone with malicious intentions. By minimizing the so called network footprint of the server, the available attack surface is also minimized. Whether or not the additional configuration overhead of preventing wildcards is worth the effort, will always remain a trade off.

- `Listen` can be used to specify the IP addresses and ports to which an Apache listener should be opened in order to serve the configured content.

The `<VirtualHost>` directive is the next step to create for each different webdomain you would like to serve. The argument to the `<VirtualHost>` directive should be the same as the argument to the (pre-Apache 2.4) `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 point out where in the filesystem the content for that webdomain can be found.

Suppose that both `www.domain.tld` and `www.otherdomain.tld` point to the IP address *`111.22.33.44`*. You could then add the following to `httpd.conf` or equivalent (included) configuration file:

```
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>
```

The IP address *`111.22.44.33`* could be replaced by `*` to match all IP addresses for this server. The implications of using wildcards in this way have been addressed above.

Many websites should be accessible by more than one name. For instance, the organization behind `domain.tld` wants to facilitate `blog.domain.tld`. There are multiple ways to implement this functionality, but one of them uses the `ServerAlias` directive. The `ServerAlias` directive is declared 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`. The DNS system must be properly configured to map those names to the IP address(es) declared 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 corresponding 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, put that configuration in a `<VirtualHost>` container and place it before any other `<VirtualHost>` container specification in the Apache configuration.

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. These are niche scenarios though:

- Some older or exotic web clients are not compatible with name-based virtual hosting for HTTP or HTTPS. HTTPS name-based virtual hosting is implemented using an extension to the TLS protocol called Server Name Indicator (SNI). Most modern browsers on modern operating systems should support SNI at the time of this writing.
- Some operating systems and network equipment devices 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 using virtual interfaces. Virtual interfaces are supported by most modern operating systems (refer to the system documentation for details on `IP aliasing` and the **ifconfig** or **ip** command).

There are two ways of running the Apache HTTP server to support multiple hosts:

- By running a separate **httpd** daemon for each hostname;
- By running a single daemon that 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 is only possible to `Listen` to the "wildcard" address, or to specific IP addresses. So, if you need to restrict one webdomain to a specific IP address, all other webdomains need to be configured to use specific IP addresses as well.

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
```

The `Listen` directive may be defined as an IP:PORT combination seperated by colons as above. Another option is to specify only the port number. By doing so, the Apache server will default to activating listeners on all configured IP addresses on the specified port(s):

```
Listen 80
Listen 443
```

The above `Listen` configuration could also be defined using `0.0.0.0` as the IP address, again using the colon as a seperator.

Another option of the `Listen` directive enables the exact specification of the protocol. In the previous example, port 80 and 443 are used. By default, Port 80 is configured for HTTP and port 443 for HTTPS in Apache. This configuration could be expanded by another HTTPS website on port 8443::

```
Listen 80
Listen 443
Listen 8443 https
```

When configuring one or more Apache daemons, the `Listen` directive may be used to specify one or more ports above 1024. This will prevent the necessity of root privileges for that daemon, if no other ports below 1025 are specified. Unless certain key or certificate files which are only accessible with root privileges are included in the configuration. You will read more about this on the next page of this book.

As of Apache 2.4, the `Listen` directive is mandatory and should be specified. Previous versions of Apache would default to port 80 for HTTP and 443 for HTTPS on all available IP addresses if no `Listen` directive was specified. Starting with Apache 2.4, the Apache server will fail to start if no valid `Listen` directive is specified.

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.snow.nl>
ServerAdmin webmaster@mail.snow.nl
DocumentRoot /groups/snow/www
ServerName www.snow.nl
ErrorLog /groups/snow/logs/error_log
TransferLog /groups/snow/logs/access_log
</VirtualHost>

<VirtualHost www.unix.nl>
ServerAdmin webmaster@mail.unix.nl
DocumentRoot /groups/unix_nl/www
ServerName www.unix.nl
ErrorLog /groups/unix_nl/logs/error_log
TransferLog /groups/unix_nl/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-URI}
```

Apache configuration for HTTPS (208.2)

Candidates should be able to configure a web server to provide HTTPS.

Key Knowledge Areas

SSL configuration files, tools and utilities

Ability to generate a server private key and CSR for a commercial CA

Ability to generate a self-signed Certificate from private CA

Ability to install the key and Certificate

Awareness of the issues with Virtual Hosting and use of SSL

Security issues in SSL use

Virtual Hosting and use of SSL through Server Name Indicator (SNI)

Disabling insecure protocols and ciphers

Terms and Utilities:

- `Apache2 configuration files`
- `/etc/ssl/, /etc/pki/`
- **openssl, CA.pl**
- `SSLEngine, SSLCertificateKeyFile, SSLCertificateFile`
- `SSLCACertificateFile, SSLCACertificatePath`
- `SSLProtocol, SSLCipherSuite, ServerTokens, ServerSignature, TraceEnable`

Resources: LinuxRef08; Engelschall00; Krause01; SSL01; SSL02; SSL03; SSL04; wikipedia_apachemodules; mozsslconf; raymii.org; cipherli.st; apachesslhowto; wikiCRIME; the **man** pages for the various commands.

Apache2 configuration files

Depending on the Linux distribution in use, the following files and directories may be used for configuration of Apache 2.x when Apache is installed from packages:

```
httpd.conf
apache.conf
apache2.conf
/etc/httpd/
/etc/httpd/conf
/etc/httpd/conf.d
/etc/apache2/
```

Configuration files are expected to contain predefined directives. If a directive is not explicitly defined, Apache will use a default setting. This default may vary per Linux distribution, so consult your distribution's Apache documentation. `/usr/share/doc` is a good place to start. Configuration files can be checked for syntax errors using either of the following commands:

```
$ sudo apachectl configtest
$ sudo service httpd configtest
```

Because Apache usually serves a daemon that listens to ports below 1024, sudo or a root shell should be used to invoke all Apache related commands. Refer to your system documentation to check for the availability of the **apachectl** or **apache2ctl** command. If both exist, they might be symlinked. The naming difference for this command has a historical reason. **apachectl** was used for Apache 1.x and when Apache2 was released the command was hence renamed match the new name. Now that Apache2.x has become the standard, either **apache2ctl** has been renamed to **apachectl** or both commands are available for compatibility reasons. When available, the **service** facility may point to httpd on Red Hat based systems, or to apache2 on Debian-based systems: The **apachectl** command has many useful options. It is in fact a shell script that functions as a wrapper for the httpd binary. Consult the man page for all available arguments and options. Just two more examples to get you started: To show all configured virtual hosts, use:

```
$ sudo apachectl -t -D DUMP_VHOSTS
```

To show all currently running websites, use:

```
$ sudo apachectl -S
```

Be careful interpreting the output from the command above. That output shows the configuration of the currently *running* websites. There is no guarantee that the website configuration on disk has changed since these websites were brought online. In other words: The output from the running processes does not necessarily have to match the contents of the configuration files (anymore).

In regards to the Apache configuration files, it is important to know about the different ways Apache may be installed and configured. Depending on the Linux distribution and Apache2.x version in use, configuration files may be located and even named differently across otherwise similar systems. As we will see further down this chapter, Apache often uses one main configuration file. Within this file, other configuration files may be included using the `INCLUDE /path/to/other/config` directive. The configuration file syntax may be checked for errors by invoking the **apachectl** script as shown previously. Each configuration file that is included from the main configuration file in use will be checked for consistency and syntax. Consistency here means that if a dependant configuration file, certificate file or key file can not be accessed properly by the user the httpd binary runs as, a warning will be shown. If **apachectl** does not appear in your **$PATH**, use the **sudo find** command with **apachectl** or **apache2ctl** as an argument. Depending on the size of your storage volumes, it may be wiser to narrow this search down to specific directories. You have been warned. If the **service** command is not available on your system, the Apache daemon may be started, checked and stopped by a SysV script instead. Look within the `/etc/init.d/` directory for a script called **httpd**, **apache2** or equivalent. This script may be then called upon as follows, to reveal the available arguments:

```
$ sudo /etc/init.d/apache2
```

Encrypted webservers: SSL

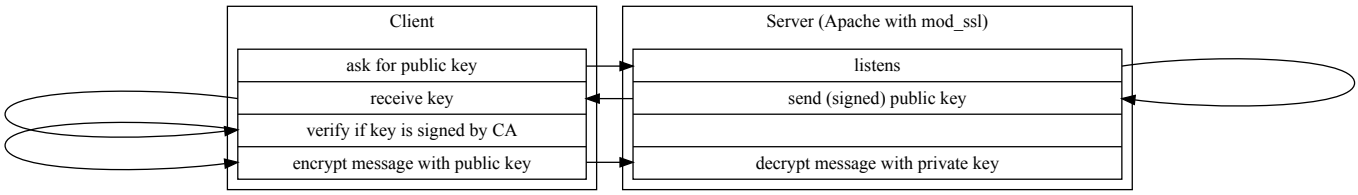
Apache can support *SSL/TLS* for (reasonably) secure online communication. While TLS in version 1.2 is actually the currently favourable option, TLS encrypted HTTPS sessions are still referred to as 'SSL' encrypted sessions. TLS could in fact be seen as the successor to SSL (v3.0). So, just as with Apache versus Apache2, whenever Apache/SSL is mentioned in this chapter, TLS is implied as well. Unless otherwise specified. We will cover the strengths and weaknesses of both protocols further down this chapter.

The Secure Sockets Layer protocol (SSL) is a protocol which may be placed between a reliable connection-oriented network layer protocol (e.g., TCP/IP) and the application layer protocol (e.g., HTTP). SSL provides secure communication between client and server by allowing mutual authentication and the use of digital signatures for integrity and encryption for privacy. Currently there are two versions of SSL still in use: version 2 and version 3. Additionally, the successor to SSL, TLS (version 1.0, 1.1 and 1.2, which are based on SSL), were designed by the IETF organisation.

Public key cryptography

SSL/TLS 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 key 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, Alice 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 using the corresponding private key.



A secure web server (e.g., Apache/SSL) uses HTTP over SSL/TLS, using port 443 by default. The SSL/TLS port can be configured by defining the **Listen** directive inside the main configuration file. There should already be a listener configured for port 80 (HTTP). On Debian-based systems, there is a dedicated file for defining the active listeners. This file is called `ports.conf` and is included from the main configuration file. Apart from this file, individual websites should specify the listening host at the end of the **ServerName** or **NameVirtualHost** declaration. Starting from Apache v2.4, **NameVirtualHost** has been deprecated in favour of **VirtualHost**. A declaration like that could look as follows: **<VirtualHost *:443>**. Within the browser, the use of HTTPS 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 should be signed (it contains a digital signature e.g., a message digest) by a so-called valid CA (Certificate Authority). Each browser contains a number of so-called *root*-certificates: these can be used to determine the validity of the CA's that signed the key. Not every certificate out there is signed by one of these valid CA's. Especially for testing purposes, it is common to sign certificates without the intervention of a valid CA. This is done in order to save both (validation) time and (registration fee) money. As of 2015, it has become easier to maintain valid CA signed certificate. An organisation called Let's Encrypt is willing to sign certificates for free, as long as you play by the rules. Use your favourite web search engine to find out more about Let's Encrypt, *after* reading this chapter.

Apache with mod_ssl

The Apache Software Foundation provides excellent documentation regarding the use of `mod_ssl`,. We urge you to take the time to read through the resources collected at the following URL: https://httpd.apache.org/docs/current/ssl/

The subject of encryption is so vast and complicated that entire books have been written around about it. The added confidentiality and integrity only provide their value when encryption is implemented correctly. So called 'best practices' in regards to encryption may change overnight. In addition to the collection of resources listed at the URL above, we want to add the following URL: http://httpd.apache.org/docs/trunk/ssl/ssl_howto.html

As you can see, this URL does not point to the *current* version of the documentation. Instead, it points to the *trunk* version. At the time of this writing, this corresponds to the Apache 2.5 documentation. The *trunk* documentation will always point towards the most recent Apache version in development. And while the *trunk* Apache code may not be recommended to use, the documentation may be more recently updated than elsewhere. In regards to the subject of SSL/TLS, this results in more up-to-date best practices than the 2.4 documentation provides.

The documentation provided by The Apache Software Foundation is vendor-neutral. So when the Apache documentation states that the following directives should be present in the Apache main configuration file:

```
LoadModule mod_ssl modules/mod_ssl.so
Listen 443
<Vhost>
</Vhost>
```

It can very well be that these directives are configured amongst several configuration files. This depends on your Linux distribution. In addition to the documentation provided by The Apache Software Foundation, we will try to point out the configuration differences between Red Hat and Debian based distributions.

To use **mod_ssl** you will need to install the Apache and mod_ssl package. On Red Hat based systems, this is done using the following command:

```
$ sudo yum install httpd mod_ssl
```

On Debian-based systems, this is done using the following command:

```
$ sudo apt-get install apache2 openssl
```

After installation, make sure the OpenSSL module is enabled within Apache. The module should be available to the Apache daemon, and included to be loaded during daemon start-up. Again, there are several ways this can be achieved. A common way is similar to the `websites-available` and `websites-enabled` strategy. However, now we are dealing with `modules-available` and `modules-enabled` directories instead. As a plus, Debian-based systems come with a utility called **a2enmod**. By invoking this command as follows:

```
$ sudo a2enmod enable ssl
```

a2enmod will create symlinks within the `mods-enabled directory, pointing to respectively mods-available ssl.conf` and `mods-available/ssl.load`. When Apache is reloaded, these symlinks will ensure the SSL module will be loaded as well.

Red Hat based systems use the **LoadModule** directive instead. This directive should be declared so it will be read during the start of the Apache daemon. On a Red Hat based system, this could be achieved by a `/etc/httpd/conf/httpd.conf` that holds the following INCLUDE directive:

```
Include conf.d/*conf
```

The default file `/etc/httpd/conf.d/ssl.conf` could then contain the following **LoadModule** and **Listen** statements:

```
LoadModule ssl_module modules/mod_ssl.so
Listen 443
```

After reloading Apache, the SSL module should be loaded together with the Apache daemon. It is always a good practice to check for configuration errors before restarting the Apache daemon. This can be done using the **apachectl configtest** command and has been covered earlier. The output should be clear to interpret whether Apache will encounter errors or not, and why (it will).

Then, generate a key and Certificate Signing Request (CSR). Either sign the csr file yourself, thus creating a 'self-signed' certificate, or have it signed by a valid Certificate Authority (CA). Depending on who you are, a self-signed certificate might cause browser-warnings when presented via HTTPS. Having the csr signed by a valid CA might prevent this from happening.

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 all of these directives. However, you should be familiar with most of the mod_ssl directives. You can find best practices by searching the web and should also refer to your distribution's specific mod_ssl documentation. The generic mod_ssl documentation can be found on the **mod_ssl** ***mod_ssl*** *web-site* .

mod_ssl can also be used to authenticate clients using client certificates. These client certificates can be signed by your own CA and **mod_ssl** will validate the certificates against this CA. To enable this functionality set the `SSLVerifyClient` to `require`. Use the value `none` to turn it off.

Certificates that are installed as part of your Linux distribution are usually installed in `/etc/ssl/certs` on Debian-based systems, and in `/etc/pki/tls/certs` on Red Hat based systems. The Red Hat based systems may have a symlink in place that points `/etc/ssl/certs` to `/etc/pki/tls/certs` for convenience and compatibility.

Keys or key-files that are installed as part of your Linux distribution are in turn usually installed in `/etc/ssl/private` on Debian-based systems and in `/etc/pki/tls/private` on Red Hat based systems. Other directories within `/etc/ssl` and `/etc/pki` may also contain specific key files.

It is often considered a best practice to create subdirectories when working with specific keys and/or certificates. Especially because specific cryptographic keys and certificates belong to each other. By devoting a dedicated subdirectory to each keypair, structure will be maintained within both the filesystem and configuration files pointing to these files. These subdirecties may be created as part of the `/etc/ssl` or `/etc/pki` hierarchy. But creating subdirectories below `/etc/apache2` or `/etc/httpd` can be done as well.

Directory /etc/ssl/

```
/etc/ssl$ ls -l
total 32
drwxr-xr-x 3 root root     16384 2011-03-06 15:31 certs
-rw-r--r-- 1 root root      9374 2010-09-24 22:05 openssl.cnf
drwx--x--- 2 root ssl-cert  4096 2011-03-06 13:19 private
```

The **openssl** program is a command line interface to the OpenSSL crypto library. You can use it to generate certificates, encrypt and decrypt files, create hashes and much more. It is generally seen as "the Swiss Army knife" of cryptography. One of the more common usages is to generate (self-signed) certificates for use on a secured webserver (to support the `https` protocol). `/etc/ssl/openssl.cnf` is the standard location for its configuration file, where you can set defaults for the name of your organization, the address etc.

Note
If you generate a certificate for a webserver you start by creating a Certificate Signing Request (.csr). The **openssl** tool will prompt you for information it needs to create the request, using defaults it fetches from the configuration file. When you generate such a signing request, make sure you enter the FQDN ("Fully Qualified Domain Name") of the server when **openssl** prompts you for the "Common Name" or CN (which is part of the "Distinguished Name"). For example when you generate a CSR for the web-site `https://www.foo.example/`, enter `www.foo.example` as the CN. Be aware that a certificate providing `foo.example` would not be valid for the website accessed via `https://www.foo.example` . Neither would this certificate be valid for the website behind the URL `https://webmail.foo.example`. Seperate certificates for each domain should be put in place. To combat this necessity, many organizations choose to use *wilcard-* certificates. Especially for internal hosted websites. When issuing a CSR for a certificate that could be used to serve any of the `.foo.example` websites, the request should be done for the CN value `*.foo.example`. Browsers will understand this wildcard certificate when presented, and decide accordingly. `www.foo.example` and `webmail.foo.example` could be configured to use this certificate. `https://foo.example` on the other hand, would issue a browser warning with this certificate.

How to create a SSL server Certificate

While installing OpenSSL, the program **openssl** is installed on your system. This command can be used to create the necessary files that implement a (self-signed) server certificate.

More specifically:

- You start by generating the *RSA key file*. It contains a pair of related keys, used to encrypt and decrypt messages to and from you. One half of the keypair will be used to encrypt messages that will be sent to you using the *public key*. The other half is used to decrypt these received messages using the *private key*. The *public key* will be made part of your *digital certificate*. This allows client systems to sent encrypted messages to your webserver that only this webserver can decrypt, as it holds the related private key;
- Next you will create a *Certificate Signing Request (CSR)*. This is a file which contains the public key and identifying information like the name of your company, location etc;
- The CSR is sent to a *Certificate Authority (CA)* which should verify the correctness of the information you provided and generate the *certificate*. This *certificate* contains a digital signature that allows verification that the CA has approved of the contents of the certificate. The certificate will contain the data you provided (including your public key) and it is signed by the CA using its private key. 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.

Note
You can create a signing request and then sign it yourself. In fact, that is what Certificate Authorities do when they create their `root certificate`. A `root certificate` is simply a certificate that says that they say they are whom they say they are. So, anybody can create a root certificate and put any credentials on it just as they please. The root certificate itself is no proof of anything. You will need to ensure that it really was issued by a party you trust yourself. Either you visit them and get a copy directly from them, or fetch it using another method you trust or you rely on others you trust to have done this for you. One of the ways you implicitly "trust" a large number of CAs is by relying on their root certificates that are made part of your browser.

As an example: to create an RSA private key that has a keysize of 2048 bits, and which will be triple-des (3DES) encrypted, stored in a file named `server.key` in the default format (which is known as PEM), type:

```
$ openssl genrsa -des3 -out server.key 2048
```

RSA keysizes below 1024 bits are considered out-of-date. 1024 bits seems to be a best practice today, with 2048, 3072, 4096 and onwards being valid options if all involved components are able to handle these keysizes without overexceeding thresholds.

openssl will ask for a pass-phrase, which will be used as the key to encrypt the private key. Please store this file in a secure backup location and remember the pass-phrase. If you loose the pass-phrase you will not be able to recover the key.

For testing purposes, it might be preferable to strip the password from the key file. This can accomplished by reading the key and exporting it as follows:

```
$ openssl rsa -in server.key -out stripped.key
```

The `server.key` file still holds the encrypted private key information in ciphertext. The `stripped.key` file is a plain text file with the unencrypted private key information as its contents. Handle with care.

To create a Certificate Signing Request (CSR) with the server RSA private key (output will be PEM formatted), execute the following:

```
$ openssl req -new -key server.key -out server.csr
```

The signing request can now either be sent to a real CA, which will sign the request and create a digital certificate, or you can create your own CA and do it yourself. Note that if you do it yourself, you will also need to install the root certificate of your CA into your clients (e.g. browser) to signal them that a certificate signed by your own CA can be trusted. If you omit this step, you will be getting a lot of disturbing warnings about missing trust and insecurity.

You can provide the **openssl** parameters yourself, but that can be a daunting task for less experienced users. Hence, for conveniences sake the OpenSSL software suite provides a perl script (**CA.pl**) to handle most CA related tasks a lot easier. It has a simplified syntax and supplies the more complex command line arguments to the underlying **openssl** command.

`CA.pl` will default use values it reads from the standard OpenSSL configuration file `/etc/ssl/openssl.cnf`. To create your own CA, find the **CA** shellscript or **CA.pl** perlscript that should be part of the OpenSSL package. On Red Hat based systems, this script is located in the `/etc/pki/tls/misc` directory. Depending on your distribution, the script might not interpret filenames for arguments. The script then instead looks for predifined values for the key file and csr file. Page the script source using a command like **less** or **more** and look for clues. The STDERR output might also show some valueable pointers. In the following example, `newkey.pem` and `newreq.pem` are used as file names by the **CA.pl** script:

```
# /usr/lib/ssl/misc/CA.pl -newca
CA certificate filename (or enter to create)

Making CA certificate ...
Generating a 2048 bit RSA private key
.......................+++
...................................+++
writing new private key to './demoCA/private/cakey.pem'
Enter PEM pass phrase: ********
Verifying - Enter PEM pass phrase: ********
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [NL]:
State or Province Name (full name) [None]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Snow B.V.]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:ssltest.snow.nl
Email Address []:
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
Serial Number:
ca:d8:22:43:94:6d:ca:6c
Validity
Not Before: Jul  9 13:49:38 2013 GMT
Not After : Jul  8 13:49:38 2016 GMT
Subject:
countryName               = NL
stateOrProvinceName       = None
organizationName          = Snow B.V.
commonName                = ssltest.snow.nl
X509v3 extensions:
X509v3 Subject Key Identifier:
83:F3:99:4B:98:E0:F1:37:78:67:DC:04:AC:04:65:03:48:BB:31:FB
X509v3 Authority Key Identifier:
keyid:83:F3:99:4B:98:E0:F1:37:78:67:DC:04:AC:04:65:03:48:BB:31:FB

X509v3 Basic Constraints:
CA:TRUE
Certificate is to be certified until Jul  8 13:49:38 2016 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
```

Next create a signing request:

```
# /usr/lib/ssl/misc/CA.pl -newreq
Generating a 2048 bit RSA private key
.........................+++
.........................+++
writing new private key to 'newkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [NL]:
State or Province Name (full name) []:None
Locality Name (eg, city) []:
Organization Name (eg, company) []:Snow B.V.
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:ssltest.snow.nl
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Request is in newreq.pem, private key is in newkey.pem
```

Then, we sign the request:

```
# /usr/lib/ssl/misc/CA.pl -signreq
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
Serial Number:
ca:d8:22:43:94:6d:ca:6d
Validity
Not Before: Jul  9 13:53:53 2013 GMT
Not After : Jul  9 13:53:53 2014 GMT
Subject:
countryName               = NL
stateOrProvinceName       = None
organizationName          = Snow B.V.
commonName                = ssltest.snow.nl
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
21:A4:61:83:B4:E7:C3:E9:2B:2C:0A:DD:36:FA:82:D0:77:3A:E2:01
X509v3 Authority Key Identifier:
keyid:83:F3:99:4B:98:E0:F1:37:78:67:DC:04:AC:04:65:03:48:BB:31:FB

Certificate is to be certified until Jul  9 13:53:53 2014 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
Signed certificate is in newcert.pem
```

You now created a certificate signed by your own CA (`newcert.pem`). You might want to rename the file to something more distinguishable, e.g `Certificate:ssltest.snow.nl`. While at it, rename the server key file too, for example `PrivateKey:ssltest.snow.nl`. Especially if you maintain a lot of keys and certificates on a lot of servers, it really helps to be able to learn from the name of a file what is in it.

The Certificate Signing Request (CSR) could have been sent to an external Certificate Authority (CA) instead. You usually have to post the CSR into a web form, pay for the signing and await a signed Certificate. There are non-profit CA's that will perform similar tasks free of charge, for example `CAcert`. However, their root certificate is not yet included into most browsers so you will need to do that yourself if you are going to use their services.

The `server.csr` file is no longer needed. Now you have two files: `server.key` and `newcert.pem`. In your Apache's `httpd.conf` file you should refer to them using lines like these:

```
SSLCertificateFile    /path/to/Certificate:ssltest.snow.nl
SSLCertificateKeyFile /path/to/PrivateKey:ssltest.snow.nl
```

It is considered a best practice to follow the 'least privilege' principle when managing key and certificate files. These files should preferebly be stored in a way that only the user account that runs the web server can access them.

Apache SSL Directives

The following Apache SSL configuration directives should be familiar to you:

SSLEngine This directive toggles the usage of the SSL/TLS Protocol Engine. This should be used inside a *<VirtualHost>* section to enable SSL/TLS for a that virtual host. By default the SSL/TLS Protocol Engine is disabled for both the main server and all configured virtual hosts.

SSLCertificateKeyFile This directive points to the PEM-encoded private key file for the server. If the contained private key is encrypted, the pass phrase dialog is forced at startup time. This directive can be used up to three times (referencing different filenames) when an RSA, a DSA, and an ECC based private key is used in parallel. For each SSLCertificateKeyFile directive, there must be a matching *SSLCertificateFile* directive.

SSLCertificateFile This directive points to a file with certificate data in PEM format. At a minimum, the file must include an end-entity (leaf) certificate. This directive can be used up to three times (referencing different filenames) when an RSA, a DSA, and an ECC based server certificate is used in parallel.

Creating and installing a self-signed certificate for Apache

Sometimes, it might be acceptable to use a self-signed SSL certificate with Apache. The following steps explain how to accomplish this on a Debian based system. First, create a directory to hold the SSL keys. On the system we use as an example, all system-wide SSL certificates are stored in the directory `/etc/ssl/certs`. For our purpose, we create a new directory called `/etc/ssl/webserver` and use it to store our new keypair:

```
# mkdir /etc/ssl/webserver
# openssl req -new -x509 -days 365 -nodes \
> -out /etc/ssl/webserver/apache.pem -keyout /etc/ssl/webserver/apache.key
Generating a 2048 bit RSA private key
.............................+++
.......+++
writing new private key to '/etc/ssl/webserver/apache.key'
# ls /etc/ssl/webserver/
apache.key  apache.pem
```

Note
During creation, openssl wil use the contents of `/etc/ssl/openssl/cnf` to fill in some variables. Other values will be asked by an interactive script. Be sure to use the proper FQDN here to distinguish this certificate from certificates with another purpose later on.

In order to be able to use SSL with Apache, a module called `mod_ssl` has to be loaded. On this system, we can check the enabled modules by listing the contents of the `/etc/apache2/mods-enabled` directory. All currently available modules can be checked by listing the contents of the `/etc/apache2/mods-available` directory:

```
# ls /etc/apache2/mods-enabled/
alias.conf            autoindex.conf  mime.conf         reqtimeout.load
alias.load            autoindex.load  mime.load         setenvif.conf
auth_basic.load       cgi.load        negotiation.conf  setenvif.load
authn_file.load       deflate.conf    negotiation.load  status.conf
authz_default.load    deflate.load    perl.load         status.load
authz_groupfile.load  dir.conf        php5.conf
authz_host.load       dir.load        php5.load
authz_user.load       env.load        reqtimeout.conf

# ls /etc/apache2/mods-available/
actions.conf          cgid.conf          include.load         proxy_ftp.conf
actions.load          cgid.load          info.conf            proxy_ftp.load
alias.conf            cgi.load           info.load            proxy_http.load
alias.load            charset_lite.load  ldap.conf            proxy.load
asis.load             dav_fs.conf        ldap.load            proxy_scgi.load
auth_basic.load       dav_fs.load        log_forensic.load    reqtimeout.conf
auth_digest.load      dav.load           mem_cache.conf       reqtimeout.load
```

```
authn_alias.load      dav_lock.load      mem_cache.load       rewrite.load
authn_anon.load       dbd.load           mime.conf            setenvif.conf
authn_dbd.load        deflate.conf       mime.load            setenvif.load
authn_dbm.load        deflate.load       mime_magic.conf      speling.load
authn_default.load    dir.conf           mime_magic.load      ssl.conf
authn_file.load       dir.load           mod-dnssd.conf       ssl.load
authnz_ldap.load      disk_cache.conf    mod-dnssd.load       status.conf
authz_dbm.load        disk_cache.load    negotiation.conf     status.load
authz_default.load    dump_io.load       negotiation.load     substitute.load
authz_groupfile.load  env.load           perl.load            suexec.load
authz_host.load       expires.load       php5.conf            unique_id.load
authz_owner.load      ext_filter.load    php5.load            userdir.conf
authz_user.load       file_cache.load    proxy_ajp.load       userdir.load
autoindex.conf        filter.load        proxy_balancer.conf  usertrack.load
autoindex.load        headers.load       proxy_balancer.load  vhost_alias.load
cache.load            ident.load         proxy.conf
cern_meta.load        imagemap.load      proxy_connect.load
```

`ssl` appears to be available but has not been enabled yet because both ssl files, `ssl.load` and `ssl.conf`, are still present in the `/etc/apache2/mods-available/` directory and not in the `/etc/apache2/mods-enabled/` directory. We could create a symlink to activate support for ssl ourselves, but Debian provides a utility written in perl called **a2enmod** that takes care of this. Consult the `A2ENMOD(8)` manpage for more information. It's counterpart, conveniently called **a2dismod**, does the opposite and disables Apache modules by removing the symlinks from `/etc/apache2/mods-enabled/`.

Let's enable SSL:

```
# a2enmod ssl
Enabling module ssl.
See /usr/share/doc/apache2.2-common/README.Debian.gz on how to configure SSL \
  and create self-signed certificates.
To activate the new configuration, you need to run:
service apache2 restart
# service apache2 restart
[ ok ] Restarting web server: apache2 ... waiting .
# apachectl status |grep -i ssl
Server Version: Apache/2.2.22 (Debian) PHP/5.4.4-15.1 mod_ssl/2.2.22 OpenSSL/
```

SSL has now been enabled on the Apache HTTP server. In order for a site to actually use SSL, it's configuration has to be properly configured. HTTPS uses tcp port 443 by default, so we want to specify this in the apache config of Debian. Add the following line to your `/etc/apache2/ports.conf` file:

```
Listen 443
```

Now, all sites that want to make use of SSL need to have their configuration files reconfigured. The following lines need to be added to each "enabled" site that should serve it's content by HTTPS:

```
SSLEngine On
SSLCertificateFile /etc/ssl/webserver/apache.pem
SSLCertificateKeyFile /etc/ssl/webserver/apache.key
```

An example site configuration file for both a HTTP and HTTPS enabled site could be like the following:

```
NameVirtualHost *:80
NameVirtualHost *:443

<VirtualHost *:80>
Servername  webserver.intranet
DocumentRoot  /srv/http
ErrorLog  /var/log/apache2/error.log
</VirtualHost>

<VirtualHost *:443>
```

```
SSLEngine On
SSLCertificateFile /etc/ssl/webserver/apache.pem
SSLCertificateKeyFile /etc/ssl/webserver/apache.key
Servername  webserver.intranet
DocumentRoot  /srv/http
ErrorLog  /var/log/apache2/error.log
</VirtualHost>
```

Now, use **apachectl configtest** to test your site configuration and if no errors occur restart the Apache HTTP server. The SSL enabled sites should now be accessible by using the `https` URL instead of `http`.

Other Apache Directives

Apart from the directives used above, the following Apache configuration directives should be familiar to you:

`SSLCACertificateFile` This directive sets the all-in-one file where you can assemble the certificates of Certification Authorities (CA) whose *clients* you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded certificate files, in order of preference.

`SSLCACertificatePath` Sets the directory where you keep the certificates of Certification Authorities (CAs) whose clients you deal with. These are used to verify the client certificate on Client Authentication.

`SSLCipherSuite` This complex directive uses a colon-separated cipher-spec string consisting of OpenSSL cipher specifications to configure the Cipher Suite the client is permitted to negotiate in the SSL handshake phase. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured Cipher Suite after the HTTP request was read but before the HTTP response is sent.

`SSLProtocol` This directive can be used to control the SSL protocol flavors mod_ssl should use when establishing its server environment. Clients then can only connect with one of the provided protocols.

`ServerSignature` The `ServerSignature` directive allows the configuration of a trailing footer line under server-generated documents (error messages, mod_proxy ftp directory listings, mod_info output, ...). The reason why you would want to enable such a footer line is that in a chain of proxies, the user often has no possibility to tell which of the chained servers actually produced a returned error message.

`ServerTokens` This directive controls whether the Server response header field which is sent back to clients includes minimal information, everything worth mentioning or somewhere in between. By default, the `ServerTokens` directive is set to `Full`. By declaring this (global) directive and setting it to `Prod`, the supplied information will be reduced to the bare minimum. During the first chapter of this subject the necessity for compiling Apache from source is mentioned. Modifying the Apache Server response header field values could be a scenario that requires modification of source code. This could very well be part of a server hardening process. As a result, the Apache server could provide different values as response header fields.

`TraceEnable` This directive overrides the behavior of `TRACE` for both the core server and mod_proxy. The default `TraceEnable on` permits `TRACE` requests per RFC 2616, which disallows any request body to accompany the request. `TraceEnable off` causes the core server and mod_proxy to return a `405` (method not allowed) error to the client. There is also the *non-compliant* setting `extended` which will allow message bodies to accompany the trace requests. This setting should only be used for debugging purposes. Despite what a security scan may say, the `TRACE` method is part of the HTTP/1.1 RFC 2616 specification and should therefore not be disabled without a specific reason.

SSL with Apache Virtual Hosts

As we saw before, the FQDN plays an important part in SSL. It has to match the CN value of the certificate. This certificate is presented to the browser when initiating the connection to the HTTPS server. Only when the certificate is valid, issued by a known, trusted and registered party, and matches the hostname will the browser iniate the connection without warnings. Otherwise, the browser should present a warning about an invalid or expired certificate, an unknown issuer, or an invalid hostname.

With IP-based virtual hosts we have a different IP/port combination for every virtual host, which means we can configure an SSL certificate for every virtual host. The HTTPS connection will be initiated on a dedicated IP address after all.

When working with name based virtual hosts however, we have no unique identifier for the resource being requested except for the hostname. So the Apache HTTP server receives all requests for the virtual hosts it serves on the same IP/port combination. It isn't until *after* the SSL connection has been established that the HTTP server knows which virtual host is actually being requested based on the URL. The URL discloses the hostname for the virtual host. But, by then it might be too late; the client could have been offered a certificate with a different CN value due to the nature of the HTTP within SSL/TLS transaction.

Currently, an extension called SNI (Server Name Indication) can be used to circumvent this name based issue. Using this extension, the browser includes the requested hostname in the first message of its SSL handshake as an UTF-8 encoded byte-string value representing the `server_name` attribute of the client hello message. This value should only consist of host and/or domain names. No IPv4 or IPv6 IP addresses should be used. Both the browser and Apache need to support SNI in order for the SNI mechanism to work. If SNI is used on the server and the browser doesn't support SNI the browser could show a "untrusted certificate" warning. This depends on the certificate that is presented for the "default" website of the HTTP server. As of this writing, most browsers do support SNI. Exceptions are the Android 2.x default browser, MS Internet Explorer on MS Windows XP before SP3 and versions of Oracle Java before 1.7 on any operating system.

To use SNI on the Apache server and prevent "untrusted certificate" warnings due to non-SNI capable browsers, it is possible to use a multidomain certificate. This certificate should contain all the necessary domain names and should be used in the Apache configuration in a separate virtual host. In this virtual host no servername should be configured and because of this it will match all requests without a hostname and therefore serving all browsers without SNI support. Apache will show the content of the right requested (SNI) site due to the requested website being extracted from the URL. This extraction takes place after the encrypted session has been created, using the multidomain certificate. This solution will probably never receive an award for it's looks, but the science behind it at least works.

Without SNI, a web server *can* serve multiple name based virtual hosts over HTTPS without browser warnings. With the requirement (or restriction) that the SSL certificate being used will be the same for all virtual hosts. The virtual hosts also have to be part of the same domain, e.g.: `virtual01.example.com` and `virtual02.example.com`. The SSL certificate has to be configured in such a way that the CN (Common Name) points to a wildcard for the domain being used, e.g.: `*.example.com`.

SSL Security Issues

The cryptographic security aspect of SSL is entirely based on trust. Mid-2013, there were about 650 Certificate Authorities. Every one of these authorities may pose as the "weakest link", and therefore the security of your SSL certificate only goes as far as your trust in it's CA.

Because;reasons (2.4.8), the `SSLCertificateChainFile` directive has been removed from the lPIC-2 objectives. This directive is good to know nevertheless. It is an addition to the `SSLCertificateFile` and `SSLCertificateKeyFile` directives. The impact of self signed certificates for HTTPS sessions has been pointed out earlier. But even if you send a CSR to a known CA and receive a validated certificate in return, set up your server correctly using this certificate and the correct key file used to generate the CSR, this does not mean every browser will validate a session using this certificate as valid. This is often due to the involved CA having signed the CSR using a certificate that is not known to your browser. This might for instance be the case if the signing certificate is newer than your browser. But also because many CA's offer different types of certificates from their product portfolio. Browsers recognize the validity of HTTPS certificates based on the availability of so called root-certificates. These could be seen as the top level within the certificate chain. The CSR's on the other hand are often signed using a so called *intermediate certificate*. These intermediate certificates are relatable to the root certificate, but exist on a lower level in the certificate chain. To restrict the amount of certificates being shipped with browser software, not all certificates being used to sign CSR's are included by default . This could result in browser warnings about an *incomplete certificate chain*. To remedy these warnings (actually errors) , one or more intermediate certificates may be needed to reassemble the certificate chain for completeness. In order to facilitate this, CA's using intermediate certificates usually offer these intermediate certificates for download. The website of the CA and/or the email holding the signed certificate information should point to the appropriate intermediate certificate(s). The different levels are often referred to as G*n*-level certificates, where *n* represents a certain number. These intermediate certicates fill the gap between your signed certificate, and the root certificates known to all major browsers. By using the `SSLCertificateChainFile` directive, you may point Apache to a file that holds two or more concatenated certificates. By concatenating the missing certificates in the right order, the certificate chain gap will be closed and the certificate chain will be complete again.

Disabling of insecure protocols and ciphers

When Apache is configured as a Secure Server to serve content via the HTTPS protocol, the client and server negotiate upon encryption initiation which *ciphers* can be used to secure the connection. Both the Apache server and the browser then offer a list of their available encryption algorithms to each other. Depending on which settings are enforced, the server and browser then initiate a secure channel using an overlapping encryption algorithm from the list of available ciphers. By setting the `SSLHonorCipherOrder` directive to a value of `on`, the server will honor the order of ciphers as specified by the `SSLCipherSuite` directive. Otherwise, the client will have the upper hand when deciding which ciphers will be used. This secure channel then is used to transmit the encryption keys that will be used to secure the communication from here on forward. Those encryption keys must also be in a cipher format that both the server and client can handle. As is often the case, trade-offs have to be made between compatibility and security. Maximizing the amount of ciphers offered (and therefore browser compatibility) also increases the possibility that one or more of the ciphers in use may be susceptible to attack vectors. The term 'weak ciphers' is often used to refer to these vulnerable encryption algorithms. The list of (de)recommended ciphers, is heavily dependant on the publication of known vulnerabilities to abuse these ciphers. This leads to opinions that may change over time. It is therefore recommended to stay up to date about known vulnerabilities. Currently, use of so called *Cipher Block Code* ciphers is not recommended. These ciphers may be identified by a "CBC" part in their name. Neither is Arcfour (or RC4 in short) a recommended cipher to be used. As far as protocols go, SSL v2 and SSL v3 are known to be vulnerable to a plethora of attack vectors. TLS v1.0 and v1.1 also have their weaknesses. TLS v1.2 is currently the recommended protocol to use if security is a concern. With TLS v1.3 being almost visible on the horizon. Apache allows for configuration of the ciphers being offered to clients. By using the following directives, the list of ciphers that Apache offers to client software can be limited:

```
SSLCipherSuite
SSLProtocol
```

The following section shows an example configuration on how these directives may be used. The `SSLCipherSuite` is configured to use strong ciphersuites only, while explicitly disabling `RC4`. The `SSLProtocol` directive disables support for `All` protocols, while explicitly enabling `TLSv1.2` support. The order in which the ciphers should be evaluated for mutual support by both the server and the client is determined by the server through use of the `SSLHonorCipherOrder` directive. Finally, the `SSLCompression` directive is configured to disable SSL/TLS compression. Disabling this compression mitigates the known CRIME (Compression Ratio Info-leak Made Easy) attack vector.

```
SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH:!RC4
SSLProtocol -All +TLSv1.2
SSLHonorCipherOrder On
SSLCompression Off
```

From Apache 2.4 on, `mod_ssl` dropped support for SSLv2. Only SSLv3 and TLS are supported. Be aware that support for recent TLS versions depends on the availability of recent OpenSSL libraries.

A good reference before configuring a secure Apache server, is the Mozilla TLS Server project at Github. This website has example configuration strings for various major HTTP servers, including Apache. The project team members should keep the example configurations up to date according to the latest recommendations. Another good reference is https://cipherli.st. This website also offers *example* configurations.

Regarding every example configuration is it always important not to copy/paste configurations without validating the content. As explained earlier, a trade-off has to be made in most cases. Configuring Apache to strictly serve a modern cipher like TLS 1.2 will mitigate most known attack vectors in regards to SSL and TLS connections. But not every browser on every operating system will be able to adhere to this requirement. The adoption of TLS v1.2 in client software requires the availability of recent SSL libraries. Not all software vendors keep these libraries on track. The trade-off here is therefore security in favor of compatibility. Using older ciphers like SSL v2 and SSL v3 will probably increase the chances of encrypted connections being vulnerable to known attacks. But at the same time this will maximize the compatibility of clients that will be able to set up these connections. Another trade off.

Apart from explicitly specifying which protocols and ciphers _can_ be used, the *preference* for using a certain protocol or cipher may vary. The order in which the directives are specified has influence, but gives no guarantees. Servers and clients often use a technique called *opportunistic encryption* to decide which of the protocols and ciphers will be used. At the same time, it is possible for client software to specify exactly what protocol and ciphers should be used. Depending on the server configuration, the server will respect these demands by the client. It is this very functionality that is the basis for a category of known attacks called *downgrade attacks*.

After having set up your server, it is regarded as a best practice to periodicly scan the system for known vulnerabilities. This can be done in multiple ways. An easy way is to use a public web service like Qualys SSLabs: https://ssllabs.com. The output will show you in detail whether and if so, which weak protocols and/or ciphers were detected.

Implementing Squid as a caching proxy (208.3)

Candidates should be able to install and configure a proxy server, including access policies, authentication and resource usage.

Key Knowledge Areas

Squid 3.x configuration files, terms and utilities

Access restriction methods

Client user authentication methods

Layout and content of ACL in the Squid configuration files

The following is a partial list of the used files, terms and utilities:

- `squid.conf`
- `acl`
- `http_access`

Resources: Kiracofe01; Brockmeier01; Wessels01; Pearson00; the **man** pages for the various commands.

Web-caches

A *web-cache*, also known as an *http proxy*, is used to reduce bandwidth demands and often allows for finer-grained access control. Using a proxy, in the client software the hostname and port number of a proxy must be specified. When the browser tries to connect to a web server, the request will be sent to the specified proxy server but to the user it looks like the request has been sent to the requested web server directly. The proxy server now makes a connection to the specified web server, waits for the answer and sends this back to the client. The proxy works like an interpreter: the client talks and listens to the proxy and the proxy talks and listens to the web server the client wants to talk to. A proxy will also use locally cached versions of web-pages if they have not yet expired and will also 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. In this case there is no need to setup a proxy in the settings of the client software. As far as the client software knows, it is talking directly to the target server, whereas it is actually 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, it caches DNS lookups, supports non-blocking DNS lookups and implements negative caching of failed requests. **squid** also 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 bandwidth saving, 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, no benefit will be gained. 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 or hosted on a slow connection, or both, you will notice 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. In most distributions all the squid 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 directory for `.conf` files. Though there is more than one file in this directory, only one file is important to most administrators, namely the `squid.conf` file. There are just about 125 option tags in this file but only eight options are really needed to get **squid** up and running. The other options just 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. However, you will need to change at least one part of the configuration file, i.e. 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 (e.g., `/usr/local/squid/cache1` for the first disk, `/usr/local/squid/cache2` for the second disk). **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 which directory the cache should be maintained. The next option is a size value in Megabytes where the default is 100 Megabytes. **squid** will store up to that amount of data in the specified directory. The next two options will set the number of subdirectories (first and second tier) to create in this directory. **squid** creates a large number of directories and stores just 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 ip addresses. The second line allows access to that range of ip 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.

`auth_param` This option is used to specify which program to start up as an 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 use 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 web sites his users can get access to. 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 or 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 an HTTP redirect to the new URL directly to the client, 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;     # Unbuffer output
while (<>) {
s@http://fromhost.com@http://tohost.org@;
print;
}
```

This Perl script replaces requests to “http://fromhost.com” with “http://tohost.org”.

Authenticators

squid can make use of authentication. 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.

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. The current supported schemes are: *basic*, *digest*, *ntlm* and *negotiate*.

Squid has some basic authentication backends. These include:

- LDAP: Uses the Lightweight Directory Access Protocol
- NCSA: Uses a NCSA-style username and password file
- MSNT: Uses a Windows NT authentication domain
- PAM: Uses the Unix Pluggable Authentication Modules scheme
- SMB: Uses a SMB server like Windows NT or Samba
- getpwam: Uses the old-fashioned Unix password file
- SASL: Uses SASL libraries (Simple Authentication and Security Layer)
- mswin_sspi: Windows native authenticator
- YP: Uses the NIS database

The *ntlm*, *negotiate* and *digest* authentication schemes provide more secure authentication methods because passwords are not exchanged over the wire or air in plain text.

Configuration of each scheme is done via the *auth_param* director in the config file. Each scheme has some global and scheme-specific configuration options. The order in which authentication schemes are presented to the client is dependent on the order the scheme first appears in the config file.

Example configuration file with multiple directors:

```
#Recommended minimum configuration per scheme:
#
#auth_param negotiate program  < uncomment and complete this line to activate>
#auth_param negotiate children 20 startup=0 idle=1
#auth_param negotiate keep_alive on
#
#auth_param ntlm program < uncomment and complete this line to activate>
#auth_param ntlm children 20 startup=0 idle=1
#auth_param ntlm keep_alive on
#
#auth_param digest program < uncomment and complete this line>
#auth_param digest children 20 startup=0 idle=1
#auth_param digest realm Squid proxy-caching web server
#auth_param digest nonce_garbage_interval 5 minutes
#auth_param digest nonce_max_duration 30 minutes
#auth_param digest nonce_max_count 50
#
#auth_param basic program < uncomment and complete this line>
#auth_param basic children 5 startup=5 idle=1
#auth_param basic realm Squid proxy-caching web server
#auth_param basic credentialsttl 2 hours
```

Access policies

Many `squid.conf` options require the use of Access Control Lists (ACLs). Each ACL consists of a name, type and value (a string or filename). ACLs are often regarded as being the most difficult part of the **squid** cache configuration, i.e. the layout and concept is not immediately obvious to most people. Additionally, the use of external authenticators and the default ACL augment 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, thereby 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 browsers access to the web-cache. The same principles apply to the other options, such as `icp_access` (Internet Cache Protocol).

To determine whether 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 requests an 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 may also require user authentication, specify an SNMP read community string, or set a TCP connection limit.

For example, these lines would keep all internal IP addresses off the Web except during lunchtime:

```
acl allowed_hosts src 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` means Wednesday until 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 192.168.1.0/255.255.255.0
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). Next you need to set up an ACL of type `proxy_auth` and add a line to regulate the access to the web-cache using that `ACL`. Here's an example:

```
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 Behaviour

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 line, 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 compared to reading 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, **squid** 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 specifies that 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`.

Implementing Nginx as a web server and a reverse proxy (208.4)

Candidates should be able to install and configure a reverse proxy server, Nginx. Basic configuration of Nginx as a HTTP server is included.

Key Knowledge Areas

Nginx

Reverse Proxy

Basic Web Server

Terms and Utilities

- `/etc/nginx/`
- **nginx**

Resources: NginX01; NginX02; the **man** pages for the various commands.

NGINX

Nginx can be used as a webserver, an HTTP reverse proxy or as an IMAP/POP3 proxy. It is pronounced as `engine-x`. Nginx is performing so well that large sites as Netflix, Wordpress and GitHub rely on Nginx. It doesn't work using threads as most of the other webserver software does but it's using an event driven (asynchronous) architecture. It has a small footprint and performs very well under load with predictable usage of resources. This predictable performance and small memory footprint makes Nginx interesting in both small and large environments. Nginx is distributed as Open Source software. There is also 'Nginx Plus', which is the commercial edition. This book will focus on the Open Source edition.

Reverse Proxy

A proxy server is a go-between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet. A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate back-end server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.

Common uses for a reverse proxy server include:

- Load balancing – A reverse proxy server can act as a "traffic cop," sitting in front of your back-end servers and distributing client requests across a group of servers in a manner that maximizes speed and capacity utilization while ensuring no server is overloaded, which can degrade performance. If a server goes down, the load balancer redirects traffic to the remaining online servers.
- Web acceleration – Reverse proxies can compress inbound and outbound data, as well as cache commonly requested content, both of which speed up the flow of traffic between clients and servers. They can also perform additional tasks such as SSL encryption to take load off of your web servers, thereby boosting their performance.
- Security and anonymity – By intercepting requests headed for your back-end servers, a reverse proxy server protects their identities and acts as an additional defense against security attacks. It also ensures that multiple servers can be accessed from a single record locater or URL regardless of the structure of your local area network.

Using Nginx as reverse HTTP proxy is not hard to configure. A very basic reverse proxy setup might look like this:

```
location / {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header Host      $host;
    proxy_pass       http://localhost:8000;
}
```

In this example all requests received by Nginx, depending on the configuration of the server parameters in `/etc/nginx/nginx.conf`, are forwarded to an HTTP server running on localhost and listening on port 8000. The Nginx configuration file looks like this:

```
server {
  listen   80;

  root /var/www/;
  index index.php index.html index.htm;

  server_name example.com www.example.com;

  location / {
    try_files $uri $uri/ /index.php;
  }

  location ~ \.php$ {
    proxy_set_header X-Real-IP  $remote_addr;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header Host $host;
    proxy_pass http://localhost:8080;
  }

  location ~ /\.ht {
    deny all;
  }
}
```

The line starting with `location ~ /\.ht` is added to prevent Nginx of displaying the content of Apache's .htaccess files. The *try_files* line is used to attempt to serve whatever page the visitor requests. If nginx is unable, then the file is passed to the proxy.

Basic Web Server

The default location for the configuration file in Nginx is `/etc/nginx/nginx.conf`.

A basic Nginx configuration, able to serve html files, looks like this:

```
server {
  # This will listen on all interfaces, you can instead choose a specific IP
  # such as listen x.x.x.x:80;
  listen 80;

  # Here you can set a server name, you can use wildcards such as *.example.com
  # however remember if you use server_name *.example.com; You'll only match subdomains
  # to match both subdomains and the main domain use both example.com and *.example.com
  server_name example.com www.example.com;

  # It is best to place the root of the server block at the server level, and not the ↩
      location level
  # any location block path will be relative to this root.
  root /usr/local/www/example.com;

  # Access and error logging. NB: Error logging cannot be turned off.
  access_log /var/log/nginx/example.access.log;
  error_log /var/log/nginx/example.error.log;

  location / {
    # Rewrite rules and other criterias can go here
    # Remember to avoid using if() where possible (http://wiki.nginx.org/IfIsEvil)
  }
}
```

For PHP support Nginx relies on a PHP fast-cgi spawner. Preferable is `php-fpm` which can be found at *http://php-fpm.org*. It has some unique features like adaptive process spawning and statistics and has the ability to start workers with different uid/gid/chroot/environment and different php.ini. The safe_mode can be replaced using this feature.

You can add the content below to `nginx.conf`. A better practice is to put the contents in a file and include this file into the main configuration file of Nginx. Create a file, for example `php.conf` and include the next line at the end of the Nginx main configuration file:

```
include php.conf;
```

The content of `php.conf` :

```
location ~ \.php {
  # for security reasons the next line is highly encouraged
  try_files $uri =404;

  fastcgi_param  QUERY_STRING       $query_string;
  fastcgi_param  REQUEST_METHOD     $request_method;
  fastcgi_param  CONTENT_TYPE       $content_type;
  fastcgi_param  CONTENT_LENGTH     $content_length;

  fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;

  # if the next line in yours still contains $document_root
  # consider switching to $request_filename provides
  # better support for directives such as alias
  fastcgi_param  SCRIPT_FILENAME    $request_filename;

  fastcgi_param  REQUEST_URI        $request_uri;
  fastcgi_param  DOCUMENT_URI       $document_uri;
  fastcgi_param  DOCUMENT_ROOT      $document_root;
  fastcgi_param  SERVER_PROTOCOL    $server_protocol;
```

```
  fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;
  fastcgi_param  SERVER_SOFTWARE    nginx;

  fastcgi_param  REMOTE_ADDR        $remote_addr;
  fastcgi_param  REMOTE_PORT        $remote_port;
  fastcgi_param  SERVER_ADDR        $server_addr;
  fastcgi_param  SERVER_PORT        $server_port;
  fastcgi_param  SERVER_NAME        $server_name;

  # If using a unix socket...
  # fastcgi_pass unix:/tmp/php5-fpm.sock;

  # If using a TCP connection...
  fastcgi_pass 127.0.0.1:9000;
}
```

Questions and answers

Web Services

1. *What is the name of the file that contains Apache configuration items that can be set on a per directory basis?*

   `.htaccess`. .htaccess [210]

2. *Which protocol is supported by the Apache web server that enables secure online communication between client en server?*

   SSL. SSL [227]

3. *What is the principle behind PKC (Public Key Cryptography)?*

   PKC is based on a public and a secret key, in which the sender of a message encrypts the data with the public key of the receiver. Only the owner of the corresponding private key can decipher this data. Public Key Cryptography [227]

4. *What is the difference between Covalent's Raven SSL Module and **mod_ssl**?*

   The difference is in the license; **mod_ssl** is open source and Covalent's Raven SSL Module is not.

5. *Which hardware component directly affects an Apache web server's performance?*

   RAM. RAM [212]

6. *Which setting controls the number of concurrent connections to an Apache web server?*

   MaxClients. MaxClients [222]

7. *Which standard format is used to write entries in the Apache web server access log file?*

   Common Log Format. Common Log Format [213]

8. *Name the two methods of access control in use by an Apache web server.*

   Discretionary access control (*DAC*) and mandatory access control (*MAC*). Access Control [214]

9. *Describe two methods of configuring authentication modules.*

   One method is to use directives in the Apache configuration files, the other is to use the **.htaccess** file. Authentication modules [217]

10. *Which command is used to restart the Apache web server?*

    apachectl restart. Apache restart [221]

11. *By which means can an Apache web server configuration file be checked for syntax errors?*

    apachectl configtest. apachectl configtest [226]

12. *Describe Apache virtual hosting.*

    Virtual hosting is a technique that provides the capability to host more than one domain on a physical host using a single web-server. Virtual Hosting

13. *Name the two methods of implementing webserver virtual hosting.*

    Name-based and *IP-based* virtual hosting. Virtual hosting methods [222]

14. *Which of the two virtual hosting methods (name or ip-based) is preferred and why?*

    Name-based Virtual Hosting, because it eases demand for scarce IP addresses. Prefered Virtual Hosting [222]

Chapter 9

File Sharing (209)

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

Objective 209.1; SAMBA Server Configuration (5 points) Candidates should be able to set up a SAMBA server for various clients. This objective includes setting up Samba as a standalone and a member server to a Windows Active Directory domain. Both setups should be configured to share directories and printers to the clients.

Objective 209.2; NFS Server Configuration (3 points) Candidates should be able to export filesystems using NFS. This objective includes access restrictions, mounting an NFS filesystem on a client and securing NFS.

SAMBA Server Configuration (209.1)

Resources: Sharpe01; the **man** pages for the various commands.

Objective 209.1; Configuring a Samba Server (5 points) Candidates should be able to set up a SAMBA server for various clients. This objective includes setting up Samba as a standalone and a member server to a Windows Active Directory domain. Both setups should be configured to share directories and printers to the clients.

Key Knowledge Areas

Samba 4 documentation

Samba configuration files

Samba tools and utilities

Mounting Samba shares on Linux

Samba daemons

Mapping Windows usernames to Linux usernames

User-Level and Share-Level security

Terms and Utilities

- **smbd, nmbd**
- **smbstatus, testparm, smbpasswd, nmblookup**
- **smbclient**

- **samba-tool**
- **net**
- `/etc/smb/`
- `/var/log/samba/`

What is Samba?

Samba implements the Server Message Block (SMB) 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 a SMB client is available can connect to the Linux machine and thus use files and printers made available by the Linux machine. Shared resources are also called "shares" or "services".

Samba is available for many platforms including Linux, AIX, HP-UX, Solaris, 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 **yum** or **rpm** (Red Hat, SuSE etc.)
- install the package using **apt-get** or **aptitude** (Debian, Ubuntu)

Samba can be run either from **inetd** or as daemons. When run via **inetd** you can save some memory and use tcpwrappers for extra security. When run as daemons, the server is always ready and sessions are faster. If you wish to use encrypted passwords, you will need to have a separate `/etc/samba/smbpasswd` file because the layout differs from `/etc/passwd`. During installation, you can choose to have `/etc/samba/smbpasswd` generated from your `/etc/passwd` file. If you choose not to do so, use **smbpasswd** to set individual passwords for users.

Samba consists of two daemons:

- **nmbd**: the NetBIOS Name Service Daemon which 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, yet another copy of **nmbd** will be running.
- **smbd**: the Server Message Block Daemon which handles file and printer access. For each client connected to the server, an extra copy of **smbd** runs.

Samba uses both the UDP and TCP protocols. TCP/139 is used for file and printer sharing. UDP is used for the registration and translation of NetBIOS names, and for browsing the network. UDP/137 is used for name service requests and responses. UDP/138 is used for datagram services to transmit small amounts of data, such as server announcements.

Samba commands

Samba core commands

smbstatus

Report on current Samba connections:

```
$ smbstatus

Samba version 4.1.12
PID     Username      Group         Machine
-------------------------------------------------------------------
23632     nobody        nobody        10.20.24.186 (ipv4:10.20.24.186:49394)

Service      pid     machine       Connected at
-------------------------------------------------------
public       23632   10.20.24.186  Sat Oct 10 10:15:11 2015

No locked files
```

testparm

Check an smb.conf configuration file for internal correctness. If **testparm** finds an error in the `smb.conf` file it returns an exit code of 1 to the calling program, else it returns an exit code of 0. This allows shell scripts to test the output from **testparm**.

Useful command line options:

-s Print service definitions without prompting for a carriage return

-v List all options; by default only the ones specified in `smb.conf` are listed

smbpasswd

Change a user's SMB password. By default (when run with no arguments) **smbpasswd** will attempt to change the current user's SMB password on the local machine. This is similar to the way the **passwd**(1) program works. When run by root it can be used to manage user accounts in the configured password backend. Please note that even though this utility is called **smbpasswd** it doesn't necessarily write the changes to the `smbpasswd` file. **smbpasswd** works on the *`passdb backend`* configured in `smb.conf`. See also Account information databases.

Command line usage:

as root: smbpasswd [options] [username]

as ordinary user: smbpasswd [options]

Useful command line options:

-a Add a new user to the password database.

-x Remove user from database

nmblookup

Is used to query NetBIOS names and map them to IP addresses in a network using NetBIOS over TCP/IP queries. The options of this command allow the name queries to be directed at a particular IP broadcast area or to a particular machine. All queries are done over UDP.

Useful command line options:

-M Search master browser.

-R Recursion. When using nmblookup to directly query a WIINS server with the UNICAST command line option recursion is needed to have the WINS server respond to queries not related to its own netbios name or IP address. Without recursion set the WINS server will only respond with its own netbios name.

-U <unicast address> Send the query to the given UNICAST address (of a WINS server) instead of broadcasting the query. Example: "nmblookup -R -U 10.10.10.2 clientname"

smbclient

Is a client that can connect to an SMB/CIFS server. It offers an interface similar to that of the ftp program (see ftp(1)). Operations include actions like getting files from the server to the local machine, putting files from the local machine to the server, retrieving directory information from the server and so on.

Useful command line options:

-L <netbios name/IP> List services available on the server responding to the given netbios name.

-I <IP address> Connect to given IP address directly instead of querying the network for the IP address of the given netbios name.

-c <command> Run given SMB command on the server. One implementation is printing with smbclient.

-U Connect as the given user.

samba-tool

Samba-tool is the main administration tool available with samba4. It can be used to configure and manage all aspects of the samba server when it is configured as an Active Directory Domain Controller (AD DC). Even though the manpages currently state otherwise, it is not supported to use **samba-tool** to configure the server as a domain member or standalone server. These options will be removed in a future version of samba-tool. Note that this tool will not be available on all systems when installed using the packages. For example, on RHEL7 and CentOS 7, it will only be available when Samba4 is installed from source.

A short list of the commands and what the are for is shown below. For a full list of the options for the commands you can view the manpage or the online manpages

dbcheck To check to local AD database for errors.

delegation To manage delegations.

dns To manage the DNS records.

domain To manage domain options, for example creating an AD DC.

drs To manage Directory Replication Services (DRS).

dsacl To manage DS ACLs.

fsmo For manage Flexible Single Master Operations (FSMO).

gpo To manage Group Policy Objects (GPO).

group To manage or create groups.

ldapcmp To compare two LDAP databases.

ntacl To manage NT ACLs.

rodc To manage Read-Only Domain Controllers (RODC)

sites To manage sites.

spn To manage Service Principal Names (SPN).

testparm To check the configuration files.

time To retrieve to time on a server.

user To manage or create users.

net

Tool for administration of Samba and remote CIFS servers. The Samba **net** utility is meant to work just like the net utility available for windows and DOS. The first argument should be used to specify the protocol to use when executing a certain command. ADS is used for ActiveDirectory, RAP is using for old (Win9x/NT3) clients and RPC can be used for NT4 and Windows. If this argument is omitted, **net** will try to determine it automatically. Not all commands are available on all protocols.

The functionality of the **net** is too extensive to cover in this section. Have a look at **man net** or **net help** to show a list of available commands and command line options. **net help <command>** will give command specific information:

```
$ net help user

net [<method>] user [misc. options] [targets]
  List users

net [<method>] user DELETE <name> [misc. options] [targets]
  Delete specified user

net [<method>] user INFO <name> [misc. options] [targets]
  List the domain groups of the specified user

net [<method>] user ADD <name> [password] [-c container] [-F user flags] [misc. options]  ↩
    [targets]
  Add specified user

net [<method>] user RENAME <oldusername> <newusername> [targets]
  Rename specified user

Valid methods: (auto-detected if not specified)
  ads       Active Directory (LDAP/Kerberos)
  rpc       DCE-RPC
  rap       RAP (older systems)

Valid targets: choose one (none defaults to localhost)
  -S or --server=<server>   server name
  -I or --ipaddress=<ipaddr>  address of target server
  -w or --workgroup=<wg>    target workgroup or domain

Valid miscellaneous options are:
  -p or --port=<port>   connection port on target
  -W or --myworkgroup=<wg>  client workgroup
  -d or --debuglevel=<level>  debug level (0-10)
  -n or --myname=<name>   client name
  -U or --user=<name>   user name
  -s or --configfile=<path> pathname of smb.conf file
  -l or --long      Display full information
  -V or --version     Print samba version information
  -P or --machine-pass    Authenticate as machine account
  -e or --encrypt     Encrypt SMB transport (UNIX extended servers only)
  -k or --kerberos    Use kerberos (active directory) authentication
  -C or --comment=<comment> descriptive comment (for add only)
  -c or --container=<container> LDAP container, defaults to cn=Users (for add in ADS only ↩
      )
```

Using **net** to get a list of shares from server "sambaserver":

```
$ net -S sambaserver -U alice share
Enter alice's password:
public
share1
share2
Printer_1
```

```
IPC$
alice
Printer_2
```

Using **net** to get the current time of server "sambaserver":

```
$ net -S sambaserver time
Sat Oct 10 10:10:04 2015
```

Commands not part of the Samba core

smbmount

NOTE: Even though **smbmount** has been abandoned by most major Linux distributions in favor of **mount.cifs** you can still expect questions about **smbmount** during your LPIC2 exam.

Even as **smbmount** was maintained by the Samba community is was not a part of the core samba-client packages. The "smbfs" package contains the **smbmount** command and must be installed to be able to use smbmount.

smbmount is used to mount file systems shared over SMB. Most probably these file systems are found on Windows systems and shared with Linux systems with SMB client software installed. **smbmount** is the command line utility for mounting SMB file systems. For a more permanent implementation the *`smbfs`* is available for use in `/etc/fstab`.

Both methods to mount SMB file systems accept options to determine how the file system is mounted. The most common options are listed here:

username Define username for authentication of the SMB session.

password Define password for authentication of the SMB session.

credentials This option points to a file containing a username and password. Use of this option is prefered over using the username and password in the command line options or in `/etc/fstab`. This file must have proper protection so only the user and/or root can read it.

```
username=value
password=value
```

uid Define UID used for the local representation of the files on the mounted file system.

gid Define GID used for the local representation of the files on the mounted file system.

fmask Define permissions of remote files in the local representation of the mounted file system. This doesn't affect the actual permissions on the remote server.

Important: The name of the option is deceptive. It's not a *mask* but the actual permissions that is defined.

dmask Define permissions of remote directories in the local representation of the mounted file system. This doesn't affect the actual permissions on the remote server.

Important: The name of the option is deceptive. It's not a *mask* but the actual permissions that is defined.

rw/ro Mount the filesystem read-write or read-only.

Example command line usage:

```
smbmount //windows/winshare2 /opt/winshare2 -o \
  username=alice.jones,password=Alice,uid=nobody,gid=nobody,fmask=775,dmask=775,rw,hard
```

Example of `/etc/fstab` usage:

```
//windows/winshare2 /opt/winshare2 smbfs \
  username=alice.jones,password=Alice,uid=nobody,gid=nobody,fmask=775,dmask=775,rw,hard ↩
      ://windows/winshare2 0 0
```

Samba logging

Samba by default writes logging to files in the `/var/log/samba/` directory:

log.nmbd Logging from the Netbios name lookup daemon.

log.smbd Logging from the SMB daemon.

Logging can be configured with global parameters in the Samba configuration. See Configuration parameters for a few of the most useful parameters.

Account information databases

Samba can be configured to use different backends to store or retrieve account information. The most important are desribed here. Smb.conf configuration option: "passwd backend".

smbpasswd

With the *smbpasswd* method a plain text file contains all account information. Passwords are encrypted.

Drawbacks to using *smbpasswd*:

- Doesn't scale.
- No replication.
- Lacks storage of Windows information (RIDs or NT groups).

Usage of *smbpasswd* is not recommended because it does not scale well or hold any Windows information.

tdbsam

Tdbsam also lacks scalability because it's just a local database (Trivial database) that doesn't support replication. One advantage of *tdbsam* over *smbpasswd* is its capabillity to also store Windows information with the accounts.

Usage of *tdbsam* is not recommended for enterprise environments because it does not scale well and holds any Windows information. Tdbsam can be used for standalone Samba servers with a recommended maximum of 250 users.

ldapsam

In enterprise environments the usage of *ldapsam* is recommended. *Ldapsam* uses LDAP as backend and LDAP is highly scalable.

Samba configuration

Samba configuration directory `/etc/smb` or `/etc/samba`.

The LPI objectives ask for knowledge about `/etc/smb/`. In some distributions `/etc/samba/` is used instead. Files and folders that exist in `/etc/smb/` or `/etc/samba/` are:

- `lmhosts` - The Samba NetBIOS hosts file;
- `smb.conf` - The configuration file for the Samba suite;
- `netlogon` - The logon directory for user logon.

smb.conf

Samba is configured via `/etc/samba/smbd.conf`. This file consists of sections containing configuration options. The name of the section is the name of the shared resource.

special sections

There are three special sections within the samba configuration file:

[global] Global Samba configuration

[homes] Special section: definition of home directories

Home directories can be accessed by using the user name as service, or by directly accessing the "homes" service:

```
smbclient //sambaserver/alice -U alice
smbclient //sambaserver/homes -U alice
```

Both these commands will result in access to the home directory of user "alice".

[printers] Special resource or service: global configuration to enable access to all printers

Note: individual printers can also be made available as a service with the *`printable`* parameter.

Configuration parameters

In this section the most important configuration options are explained, grouped by Samba configuration section (type). Most can also be found in the examples section further on.

The smb.conf man pages divide parameters into two groups:

global Parameters that can only be used in the *`[global]`* sections of the Samba configuration.

services Parameters that are used in service sections of the Samba configuration.

Some of these paramaters can also be used globally.

[global]

The *`[global]`* section contains global parameters, but it is also used to set service parameters in a global context (providing default values if the parameter is not set for a specific service).

netbios name This option sets the NetBIOS name by which the Samba server is known. This name will be the name that services are advertised under. By default it is the same as the system's hostname.

netbios aliases This option sets an alias by which the Samba server is alternatively known.

log file This option dictates to what file(s) logging is written. The file name accepts macros enabling for instance writting a log file per client: *`/var/log/samba/log.%m`*.

workgroup Server and clients must be members of the same workgroup.

realm This option specifies the kerberos realm to use. The realm is used as the ADS equivalent of the NT4 domain.

server string Any string you want to apear in list contexts.

encrypt passwords Windows encrypts passwords. This option will also need to be turned on for Samba.

security This option determines what security mode to use. Most commonly used is `user` for standalone file servers or Samba servers that also function as a DC. If the Samba server is connected to a Windows domain this option must be set to `ads` or `domain`.

unix password sync This boolean parameter in the *[global]* section controls whether Samba attempts to synchronize the UNIX password with the SMB password when the encrypted SMB password in the `smbpasswd` file is changed. If this is set to yes (unix password sync = yes), the program specified in the **passwd** program parameter is called AS ROOT - to allow the new UNIX password to be set without access to the old UNIX password (as the SMB password change code has no access to the old cleartext password).

passdb backend This option determines what account/password backend is used. See also Account information databases

Mostly used options are:

smbpasswd[:argument] Old plaintext passdb backend. Optionally takes a path to the smbpasswd file as an argument.
Example: `passdb backend =smbpasswd:/etc/samba/smbpasswd`

tdbsam[:argument] TDB based password storage backend. Optionally takes a path to the TDB file as an argument.
Example: `passdb backend =tdbsam:/etc/samba/private/passdb.tdb`

ldapsam[:argument] LDAP backend. Optionally takes an LDAP URL as an argument. (defaults to "ldap://localhost")
Example: `passdb backend =ldapsam:ldap://localhost`

username map This option in the *[global]* section allows you to map the client supplied username to another username on the server. The most common usage is to map usernames used on DOS or Windows machines to those used on the UNIX system. Another usage is to map multiple users to a single username so that they can more easily share files. The username map is a file where each line should contain a single UNIX username on the left then a "=" followed by a space-delimited list of usernames on the right. Quotes must be used to specify a username that includes a space. The list of usernames on the right may contain names of the form @group in which case they will match any UNIX username in that group. The special client name "*" is a wildcard and can be used to match unknown names to a known user. Each line of the map file may be up to 1023 characters long. If a line begins with a "!" then the processing will stop at that point if it matches a name. This is useful for lines used before a wildcard otherwise names will still be mapped to the one using the wildcard.

Here is an example:

```
username map = /usr/local/samba/private/usermap.txt
```

Example content of `usermap.txt`:

```
root = administraor admin
nobody = guest pcguest smbguest
alice.jones = alice
readonly = glen fred terry sarah
lachlan = "Lachlan Smith"
users = @sales
```

guest ok This parameter configures guest access for a service.

map to guest Is guest access is enabled this option determines what sessions are mapped to guest access. Available values are:

- Never
- Bad User
- Bad Password
- Bad Uid (only available in ADS or DOMAIN security mode)

service sections

The following parameters are used in service definitions (both special as normal services).

path The context in which this parameter is used determines how it is interpreted:

- In the *[homes]* section it specifies the path to the directory that must be served as the users home directories. If omitted the home directory defaults to the system's home directory. If used this parameter must contain the "%S" macro, expanding to the username.

- In a section that is set to be `printable` this parameter points to the directory where printer spool files are written prior to being sent to the print queue. This directory must be world-writable and have the sticky bit set if the printer is configured for guest access.
- In a `share` definition this parameter points to the directory the share must give access to.

comment Text field showing in service listings.

printer name Points to a local print queue when configuring an individual printer.

printable Declares a service as a printer.

browseable Makes service browseable. A client can browse to a service instead of having to know the full path to the service.

guest ok Guest access is enabled for this service (or globally).

(in)valid users Provide a list of users that are allowed access (`valid users`) to this service, or that are denied access (`invalid users`). Names starting with a "@" are interpreted as a NIS netgroup or a Unix group. When a name starts with a "+" the nsswitch mechanism is used to find the group. With a "&" the group will only bee looked up in NIS. See the manual for more information.

hosts allow|deny Provide a list of clients that can be granted (`hosts allow`) or denied (`hosts deny`) access. Names can be IP addresses, networks or host names. Names started with a "@" are NIS netgroups.

writable Determines if a user is allowed to write to this service. Defaults to "no".

Security levels and modes

Samba knows two security levels: "user-level" and "share-level". The server will inform the client of the security level and the client will respond in correspondance with the choosen level. The security *level* is determined by setting the security *mode*.

The security mode is a Global setting.

User-level security

User-level security means that each connection is authenticated by a username and password combination which has to match with authorizations on the requested service. For "user-level" security the server can be set up in three modes:

user `security =user`

Samba is running as a standalone server and will use a local password database.

ads `security =ads`

Samba will act as a Active Directory domain member in an ADS realm.

domain `security =domain`

Samba will validate the username/password to a Windows NT Primary or Backup Domain Controller.

Share-level security

`security =share`

With share-level security the client expects a password to be associated with each share, independent of the user. With share-level security the client will only pass the password provided by the remote user and does so for each seperate share. The Samba sever will then try to match the password to a confgured list of users (if provided for the share that's affected), or will use system calls (looking in nsswitch.conf or /etc/passwd) to find a Linux account matching the provided password.

Share-level service parameters:

only user *only user -yes*

Only the users listed in *username* have access to this service. If not any user matching the provided password is given access.

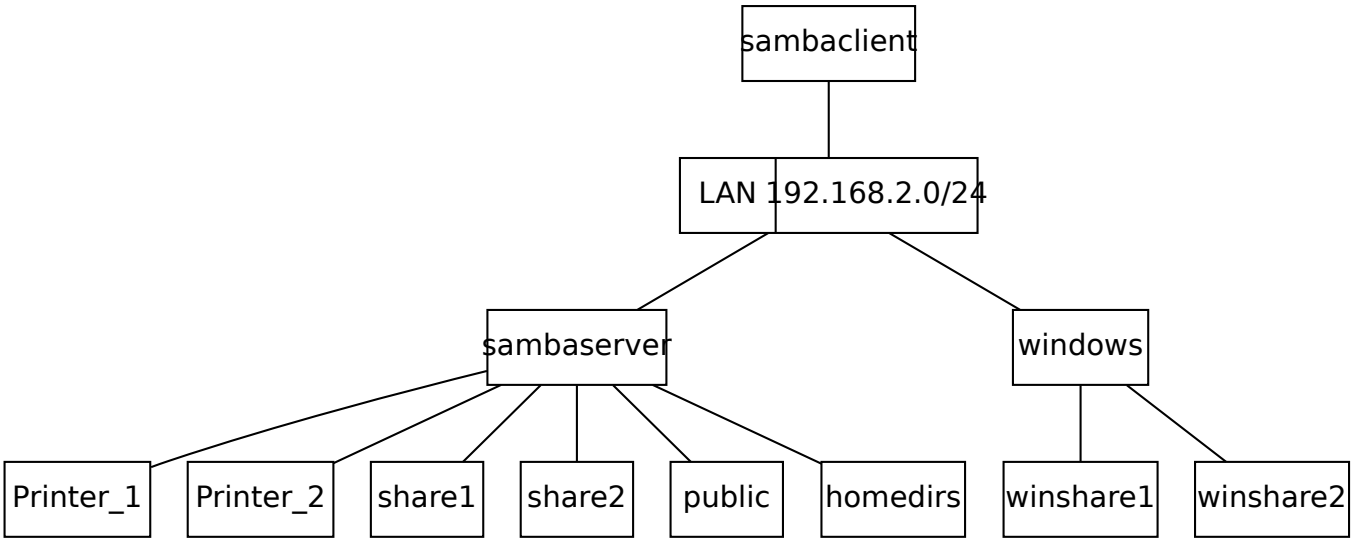
username *username =fred, alice*

Determines which users have access to this service.

Note: Because with share-level security the password to access a share is not known by just one person but by everyone who needs access share-level security is considered to be insecure and therefor support for share-level security has been removed from Samba version 4.

Examples

The following image decribed the environment used to implement the examples described below.



We've got three machines connected via a network on which we want to accomplish the following:

- The machines "sambaserver" and "windows" contain files the other machines must be able to manipulate.
- All machines must be able to use the printer connected to "sambaserver".
- "sambaserver" is running Linux and has Samba installed.
- "windows" is running Microsoft Windows.
- "sambaclient" is running Linux and has **smbclient** installed to be able to function as a Samba client.
- We want to share only parts of the resources on "sambaserver" and "windows".
  - Make share public available to everyone
  - Make share share1 available to alice
  - Make share share2 available to authenticated users
  - Make the home directories available to their respective owners
  - Map remote user alice.jones to user alice
  - Make shares on windows available to users on sambaclient
  - Allow everyone to print on all printers on sambaserver
  - Disallow printing on Printer_1 from sambaclient
  - List available services on sambaserver

Basic [global] section to support the examples

Basic global section needed to support the following examples:

```
[global]
  workgroup = OURGROUP
  server string = Linux Samba Server %L for LPIC2 examples
  encrypt passwords = yes
  security = user
  netbios name = sambaserver
  netbios aliases = ss2
  log file = /var/log/samba/log.%m
  map to guest = bad user
  hosts allow =
  valid users =
  guest ok = no
```

Example: Make "public" share available to everyone

The configuration section inserted or modified to implement this example:

```
[public]
  comment = Public Storage on %L
  path = /export/public
  browsable = yes
  writeable = yes
  guest ok = yes
  # valid users =
```

- The section *[public]* is added.
- The path that is made accessible by this service is `/export/public`.
- The service is made browsable so a client can browse to the service by connecting directly to the Samba server.
- The service is made writable.
- Guest access is enabled so no authentication is needed.
- *valid users* is not set and the global value is used (defaults to "all authenticated users"): all authenticated users have access.

All authenticated users have access and users that can not be authenticated will get access as "guest".

Create a test file, connect with account "jack" that cannot be authenticated (effectively a "guest"), check the active share and copy the test file to the share.

```
$ touch jack.txt
$ smbclient //SAMBASERVER/public -U jack -N
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]
smb: \> volume
Volume: |public| serial number 0x2b5c2e91
smb: \> put jack.txt
putting file jack.txt as \jack.txt (0.0 kb/s) (average 0.0 kb/s)
smb: \> ls
  .                                   D        0  Wed Oct 21 09:16:57 2015
  ..                                  D        0  Wed Oct 21 07:44:56 2015
  public.txt                          N        0  Wed Oct 21 07:45:07 2015
  jack.txt                            A        0  Wed Oct 21 09:16:57 2015

    54864 blocks of size 131072. 47234 blocks available
smb: \>
```

Output of smbstatus showing the session from user “nobody” which is our (default) configured Linux account for “guest” and checking the test file on the “public” share:

```
$ smbstatus

Samba version 4.1.12
PID     Username      Group         Machine
-------------------------------------------------------------------
24265     nobody        nobody        10.20.27.158 (ipv4:10.20.27.158:49009)

Service      pid     machine       Connected at
-------------------------------------------------------
public       24265   10.20.27.158  Wed Oct 21 08:58:48 2015

No locked files
$ pwd
/export/public
$ ls -l
total 0
-rwxr--r--. 1 nobody nobody 0 Oct 21 09:16 jack.txt
-rw-r--r--. 1 root   root   0 Oct 21 07:45 public.txt
```

Example: Make “share1” share available to alice

The configuration section inserted or modified to implement this example:

```
[share1]
  comment = Share1 on %L
  path = /export/share1
  # guest ok = no
  browsable = yes
  writeable = yes
  valid users = alice
```

- The section *[share1]* is added.
- The path that is made accessible by this service is `/export/share2`.
- The service is made browsable so a user can browse to the service by directly connecting to the Samba server.
- The service is made writable.
- *guest ok* is not set and the global value is used (default = “no”): guest access is not allowed.
- *valid users* is set to “alice” to allow access for the *Linux* user “alice”.

Failing attempt to access share1 as fred:

```
$ smbclient //SAMBASERVER/share1
Enter fred’s password:
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]
tree connect failed: NT_STATUS_ACCESS_DENIED
```

Successful attempt to access share1 as alice:

```
$ smbclient //SAMBASERVER/share1
Enter alice’s password:
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]
smb: \> volume
Volume: |share1| serial number 0xd62d5fc5
smb: \>
```

Example: Make “share2” share available to authenticated users

The configuration section inserted or modified to implement this example:

```
[share2]
  comment = %S on %L
  path = /export/share2
  browsable = yes
  writeable = no
  # guest ok = no
  # valid users =
```

- The section *[share2]* is added.
- The path that is made accessible by this service is `/export/share2`.
- The service is made browsable so a user can browse to the service by directly connecting to the Samba server.
- The service is NOT writable.
- *guest ok* is not set and the global value is used (default = “no”): guest access is not allowed.
- *valid users* is not set and the global value is used (default = “empty”): all authenticated users have access.

Because *guest ok* defaults to the global value of “no” and the empty *valid users* defaults to the global value of “ any authenticated user” all (and only) authenticated users have access to “share2”

Failing attempt to access share2 as guest:

```
$ smbclient //SAMBASERVER/share1
Enter jack’s password:
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]
tree connect failed: NT_STATUS_ACCESS_DENIED
```

Successful attempt to access share2 as an authenticated user:

```
$ smbclient //SAMBASERVER/share2
Enter alice’s password:
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]
smb: \> volume
Volume: |share2| serial number 0xb954cdf0
smb: \>
```

Example: Make the home directories available to their respective owners

The configuration section inserted or modified to implement this example:

```
[homes]
  comment = %U’s homedirectory on %L from %m
  # path =
  browsable = no
  writeable = yes
  # guest ok = no
  # valid users =
```

- The section *[public]* is added.
- The path that is made accessible by this service is `/export/public`.
- The service is made browsable so a user can browse to the service by directly connecting to the Samba server.

- The service is made writable.
- Guest access is enabled so no authentication is needed.
- *valid users* is not set and the global value is used (default = “empty”): all authenticated users have access to this special service.

As “fred” access your home directory on “sambaserver”:

```
$ smbclient //SAMBASERVER/fred
Enter fred’s password:
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]
smb: \> volume
Volume: |fred| serial number 0xce0909dd
smb: \>
```

Output of smbstatus showing the session from user “fred”:

```
$ smbstatus
Samba version 4.1.12
PID     Username      Group         Machine
-------------------------------------------------------------------
24457     fred          fred          10.20.27.158 (ipv4:10.20.27.158:49017)

Service      pid     machine       Connected at
-------------------------------------------------------
fred         24457   10.20.27.158  Wed Oct 21 09:36:34 2015

No locked files
```

Example: Map remote user “alice.jones” to Linux user “alice”

Parameter added to the *global* section:

```
[global]
  ...
  username map = /etc/samba/usermap.txt
  ...
```

Sample contents of `/usr/local/samba/private/usermap.txt`:

```
root = administraor admin
nobody = guest pcguest smbguest
alice = alice.jones
readonly = glen fred terry sarah
lachlan = "Lachlan Smith"
users = @sales
```

- User mapping is a global setting. Login names (most probably Windows account names) are mapped to local (Linux) users.

If “alice.jones” tries to connect to the related home directory “alice.jones” will be mapped to “alice”, the user will have access to all services enabled for “alice” and the home directory for “alice” will be served instead of “alice.jones”.

Connection from “sambaclient” to “sambaserver” as “alice.jones”:

```
$ smbclient //SAMBASERVER/alice.jones
Enter alice.jones’s password:
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]
smb: \> volume
Volume: |alice| serial number 0x37da1047
smb: \>
```

Output of **smbstatus** on “sambaserver” showing active connections doesn’t show “alice.jones” but only “alice”:

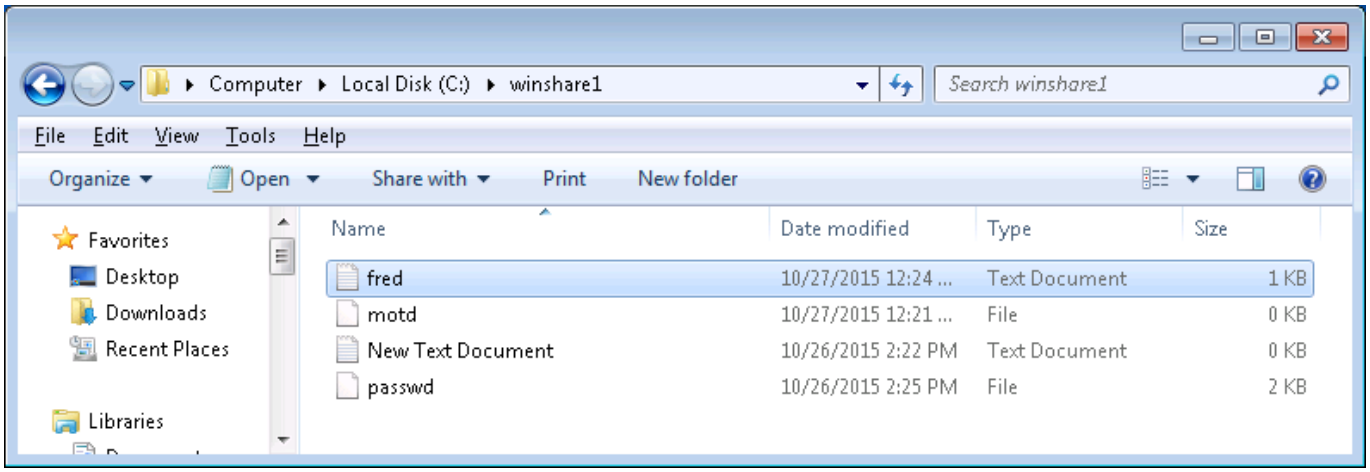
```
$ smbstatus

Samba version 4.1.12
PID     Username      Group         Machine
-------------------------------------------------------------------
23788     alice         alice         10.20.27.158 (ipv4:10.20.27.158:48988)

Service      pid     machine       Connected at
-------------------------------------------------------
alice        23788   10.20.27.158  Wed Oct 21 07:29:39 2015

No locked files
```

Example: Make shares on “windows” available to users on “sambaclient”

Using **smbclient** to copy a file to `winshare1` on “windows”:

```
[fred@sambaclient ~]$ echo "file from Fred" > fred.txt
[fred@sambaclient ~]$ smbclient //windows/winshare1
Enter fred’s password:
Domain=[WINDOWS] OS=[Windows 7 Professional 7601 Service Pack 1] Server=[Windows 7  ↩
    Professional 6.1]
smb: \> dir
  .                                   DR        0  Tue Oct 27 07:21:20 2015
  ..                                  DR        0  Tue Oct 27 07:21:20 2015
  desktop.ini                        AHS       46  Tue Oct 27 07:16:15 2015
  motd                                 A        0  Tue Oct 27 07:21:20 2015
  New Text Document.txt                A        0  Mon Oct 26 09:22:17 2015
  passwd                               A     1055  Mon Oct 26 09:25:34 2015

    40551 blocks of size 262144. 3397 blocks available
smb: \> put fred.txt
putting file fred.txt as \fred.txt (14.6 kb/s) (average 14.6 kb/s)
smb: \> dir
  .                                   DR        0  Tue Oct 27 07:23:58 2015
  ..                                  DR        0  Tue Oct 27 07:23:58 2015
  desktop.ini                        AHS       46  Tue Oct 27 07:16:15 2015
  fred.txt                             A       15  Tue Oct 27 07:23:58 2015
  motd                                 A        0  Tue Oct 27 07:21:20 2015
  New Text Document.txt                A        0  Mon Oct 26 09:22:17 2015
  passwd                               A     1055  Mon Oct 26 09:25:34 2015

    40551 blocks of size 262144. 3397 blocks available
```

Checking the result on “windows”:



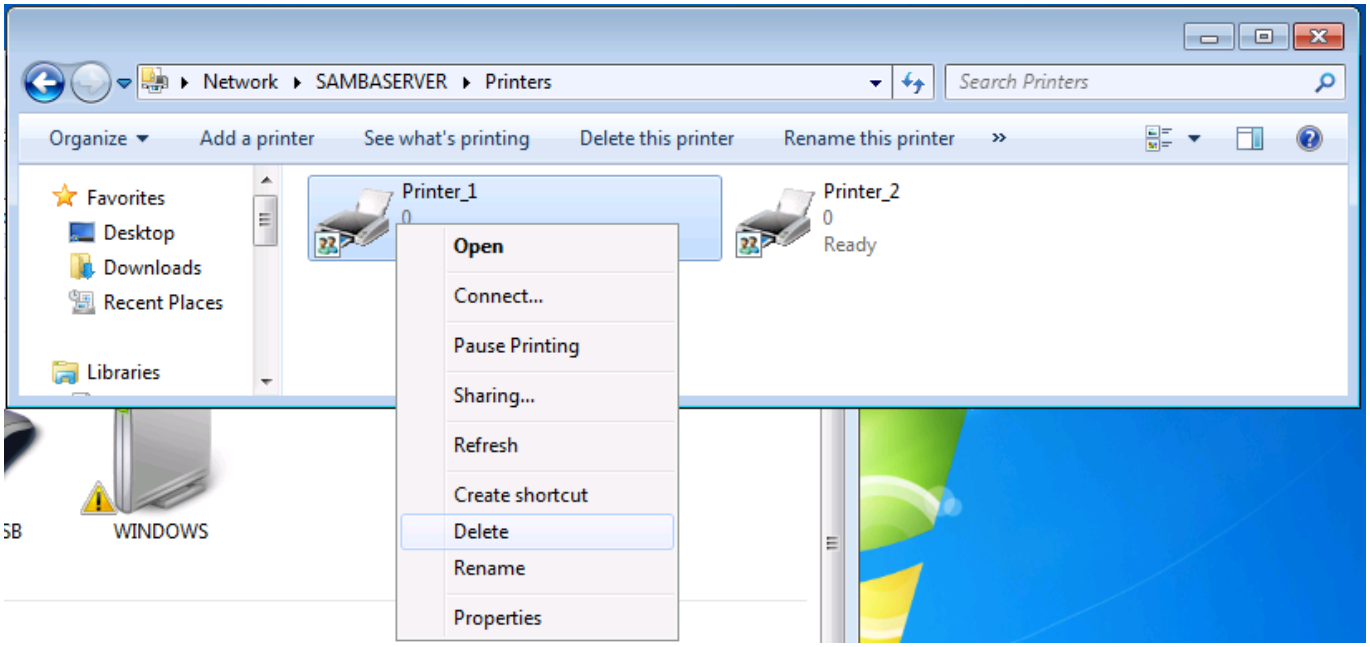
Example: Allow everyone to print on all printers on "sambaserver"

The configuration section inserted or modified to implement this example:

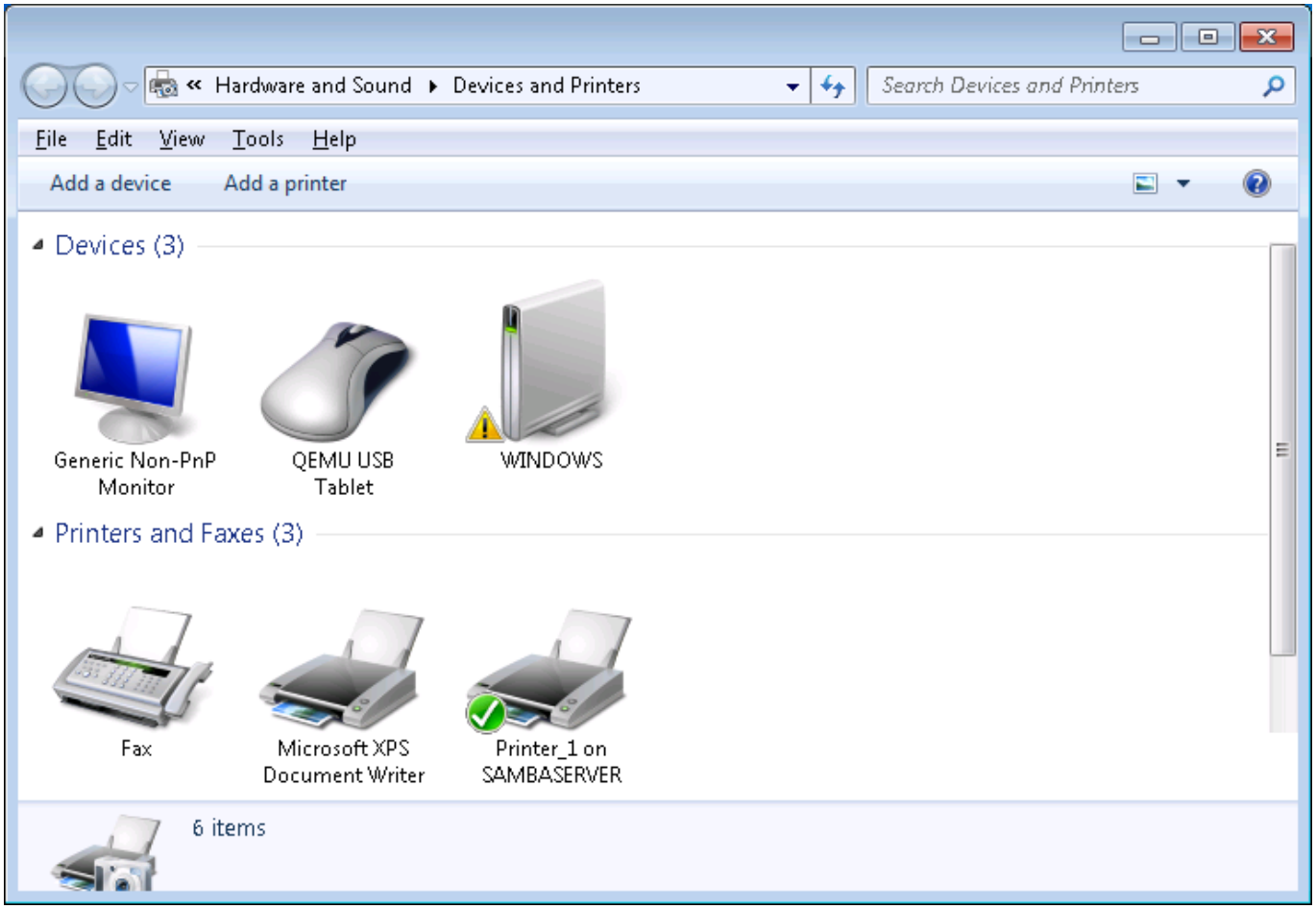
```
[printers]
  comment = Printer %p on %L
  path = /var/spool/samba
  printable = yes
  browseable = yes
  guest ok = yes
  # valid users = #
```

- The special section `[printers]` is added.
- Spool files are written to `/var/spool/samba`.
- The services matching this section (all printers) are made printable.
- The service is made browsable so it can be looked up by connecting to the server.
- Guest access is enabled so no authentication is needed.

Using Windows Explorer on Windows to browse and connnect to (enable) printer_1.



Right click enables connecting to the printer and adding it as a Generic text based printer.



Right click Printer_1 and print test page. Checking the spool file of the printer on “sambaserver”:

```
                                Windows
                            Printer Test Page
Congratulations!
If you can read this information, you have correctly installed your
Generic / Text Only on WINDOWS.
The information below describes your printer driver and port settings.
Submitted Time: 11:45:31 AM .10/.26/.2015
Computer name:  WINDOWS
Printer name:   \\SAMBASERVER\Printer_1
Printer model:  Generic / Text Only
Color support:  No
Port name(s):   \\SAMBASERVER\Printer_1
Data format:    RAW
Driver name:    UNIDRV.DLL
Data file:      TTY.GPD
Config file:    UNIDRVUI.DLL
Help file:      UNIDRV.HLP
Driver version: 6.00
Environment:    Windows NT x86
Additional files used by this driver:
 C:\Windows\system32\spool\DRIVERS\W32X86\3\TTYRES.DLL
(6.1.7600.16385 (win7_rtm.090713-1255))
 C:\Windows\system32\spool\DRIVERS\W32X86\3\TTY.INI
 C:\Windows\system32\spool\DRIVERS\W32X86\3\TTY.DLL
(6.1.7600.16385 (win7_rtm.090713-1255))
 C:\Windows\system32\spool\DRIVERS\W32X86\3\TTYUI.DLL
```

```
(6.1.7600.16385 (win7_rtm.090713-1255))
 C:\Windows\system32\spool\DRIVERS\W32X86\3\TTYUI.HLP
 C:\Windows\system32\spool\DRIVERS\W32X86\3\UNIRES.DLL
(6.1.7600.16385 (win7_rtm.090713-1255))
 C:\Windows\system32\spool\DRIVERS\W32X86\3\STDNAMES.GPD
 C:\Windows\system32\spool\DRIVERS\W32X86\3\STDDTYPE.GDL
 C:\Windows\system32\spool\DRIVERS\W32X86\3\STDSCHEM.GDL
 C:\Windows\system32\spool\DRIVERS\W32X86\3\STDSCHMX.GDL
This is the end of the printer test page.
```

Example: Disallow printing on “Printer_1” from “sambaclient”.

The configuration section inserted or modified to implement this example:

```
[Printer_1]
  comment = Printer 1 on %L
  path = /var/spool/samba
  printer name = Printer_1
  printable = yes
  browseable = yes
  guest ok = yes
  hosts deny = sambaclient
```

- A section is created to explicitely match “Printer_1”
- Making the service printable identifies the service as a printer
- Spool files are written to `/var/spool/samba`
- Print jobs are sent to the local printer queue “Printer_1”
- The service is not made browseable, so cannot be looked up by connecting to the server
- Guest access is enabled so no authentication is needed.
- Access is explicitely denied for “sambaclient”.

Samba will first match the requested service against sections explicitely matching the service name before trying a match on the special section *`[printers]`*. Any service other than “Printer_1” will not match any explicite sections and will fall through to the special section *`[printers]`*. A request for service “Printer_1” will first match the *`[Printer_1]`* section and will therefor never match the special section *`[printers]`*.

Please note that in Samba it is not possible to configure something like “sambaclient has access to all printer except for Printer_1”. In this case we need to configure all printers to accessible for all and add a configuration for any exception.

Using **smbclient** to test printing from sambaclient

```
$ smbclient //sambaserver/Printer_1/ -c "print /etc/hosts"
Enter alice’s password:
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]
tree connect failed: NT_STATUS_ACCESS_DENIED
```

Example: List available services on “sambaserver”.

This example doesn’t need additional configuration.

Using **smbclient** to create a listing of “sambaserver”. Note the comments.

```
$ smbclient -L //SAMBASERVER
Enter fred's password:
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]

  Sharename       Type      Comment
  ---------       ----      -------
  public          Disk      Public Storage on sambaserver
  share1          Disk      Share1 on sambaserver
  share2          Disk      share2 on sambaserver
  Printer_1       Printer   Printer 1 on sambaserver
  IPC$            IPC       IPC Service (Linux Samba Server sambaserver for LPIC2  ↩
      examples)
  fred            Disk      fred's homedirectory on sambaserver from sambaclient
  Printer_2       Printer   Cups printer Printer_2
Domain=[OURGROUP] OS=[Unix] Server=[Samba 4.1.12]

  Server               Comment
  ---------            -------
  SAMBACLIENT          Samba 4.1.12
  SAMBASERVER          Linux Samba Server sambaserver for LPIC2 examples
  SS2                  Linux Samba Server sambaserver for LPIC2 examples

  Workgroup            Master
  ---------            -------
  OURGROUP             SAMBASERVER
```

Setting up a 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. This 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".

Restart the smb and nmb services to pick up the changed configuration

```
# service smb restart
# service nmb restart
```

Creating logon scripts for clients

Logon scripts can be very handy. 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 "fred", create a file called `/home/netlogon/fred.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 "workstation1", create a file called `/home/netlogon/workstation1.bat`.

Configuring Samba as a domain member

To configure Samba4 as a domain member you need to make sure there is no configuration present on the system before starting.

There are two options for joining a domain. The server can be a member of and Active Directory domain or an older NT4 domain. Because an Active Directory domain uses Kerberos and DNS it is importatnt to configure the server correctly before joining the domain.

Configuring DNS

For the server to locate the domain it is important that the DNS settings are configured correctly. An AD DC has a built-in DNS server which should be used by the system we want to connect. When manually configuring the ip settings you should configure the AD Domain Controller as the DNS server. How you do this depends on the distribution you use.

When configured correctly you `/etc/resolv.conf` file should look as follows when the AD Domain Controller has an ipaddres of 192.168.1.2 and the domain is example.com:

```
nameserver 192.168.1.2
search example.com
```

When you join the host to the domain Samba tries to register the host in the AD DNS zone. For this the **net** utility tries to resolve the hostname using DNS or a correct entry in /etc/hosts.

When using `/etc/hosts` it is important that the hostname or FQDN doesn't resolve to 127.0.0.1. Because of this, a correctly configured hostfile will look as follows where server2.example.com is the hostname of the server we are adding as a domain member:

```
127.0.0.1   localhost localhost.localdomain
192.168.1.3 server2.example.com server2
```

To check if the resolution is correct you can use the **getent** command as follows:

```
$ getent hosts server2
192.168.1.3 server2.example.com server2
```

Configuring Kerberos

Currently Samba uses Heimdal Kerberos. This means that the Kerberos file `/etc/krb5.conf` only needs to contain the following information:

```
[libdefaults]
    default_realm = EXAMPLE.COM
    dns_lookup_realm = false
    dns_lookup_kdc = true
```

Using anyting other than the above can lead to errors.

You will need to replace EXAMPLE.COM with you Kerberos realm.

Kerberos requires a synchronised time on all domain members. It is recommended to set up a NTP client.

Configuring Samba

The previous steps are only necessary when joining an Active Directory domain. The following steps are needed for both an Active Directory domain and a NT4 domain.

Setting up the smb.conf file

The next step is to configure the domain members `smb.conf` file. This file is usually located at `/etc/smb/smb.conf` or `/etc/samba/smb.conf`. If not you can use the following command to locate the file:

```
$ smbd -b | grep CONFIGFILE
CONFIGFILE: /usr/local/samba/etc/smb.conf
```

Now that we know where the file is located we can add the following configuration:

```
[global]
    security = ADS
    workgroup = EXAMPLE
    realm = EXAMPLE.COM

    log file = /var/log/samba/%m.log
    log level = 1

    # Default ID mapping configuration for local BUILTIN accounts
    # and groups on a domain member. The default (*) domain:
    # - must not overlap with any domain ID mapping configuration!
    # - must use a read-write-enabled back end, such as tdb.
    idmap config * : backend = tdb
    idmap config * : range = 3000-7999
```

Joining the domain

Now that we have configured samba it's time to join the domain. As also stated above it's not supported to use the **samba-tool** utility to do this.

To join a domain you can use the following command. The output will depend on the type of domain you're joining.

When joining an Active Directory domain:

```
$ net ads join -U administrator
Enter administrator's password:
Using short domain name - EXAMPLE
Joined 'server2' to dns domain 'example.com'
```

When joining a NT4 domain:

```
$ net ads join -U administrator
Enter administrator's password:
Joined domain EXAMPLE.
```

Configuring the Name Service Switch (NSS)

To make the domain users and groups available to the local system we have to append the winbind entry to the following databases in `/etc/nsswitch.conf`:

```
passwd: files winbind
group: files winbind
```

Starting the services

Now we can start the services. If you only need Samba to lookup domain users and groups you only have to start the **winbind** service. If you also set up file and printer sharing you also need to start the **smbd** and **nmbd** services.

```
$ systemctl start winbind smbd nmbd
```

You should NOT start the **samba** service. This service is only required on Active Directory Domain Controllers,

Testing the winbind connectivity

To verify if the winbind service is able to connect to Active Directory Domain Controllers or NT4 Domain Controllers you can use the **wbinfo** command:

```
$ wbinfo --ping-dc
Checking the NETLOGON for domain[EXAMPLE] dc connection to "server1.example.com" ↩
    succeeded
```

Configuring an NFS Server (209.2)

Resources and further reading: NFS, NFSv4, NFSv4.2, Zadok01.

Candidates should be able to export filesystems using NFS. This objective includes access restrictions, mounting an NFS filesystem on a client and securing NFS.

Key Knowledge Areas

NFS version 3 configuration files

NFS tools and utilities

Access restrictions to certain hosts and/or subnets

Mount options on server and client

TCP Wrappers

Awareness of NFSv4

Terms and Utilities

- `/etc/exports`
- **exportfs**
- **showmount**
- **nfsstat**
- `/proc/mounts`
- `/etc/fstab`
- **rpcinfo**
- **mountd**
- **portmapper**

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 being adjusted, 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 Section 9.2.8 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 point 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 See Kernel options for NFS.

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*.

Kernel options for NFS provides an overview of NFS support in the kernel.

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 used to interface TCP/IP connections to the appropriate RPC calls. It is needed for all NFS traffic [1] because not only does it map (incoming) TCP connections to NFS (RPC) calls, it also can be used to map different NFS versions to different ports on which NFS is running. 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.

[1] Strictly speaking, you can run NFS without a portmapper on a client system, however, connections will be slow and (even more) unreliable.

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

Table 9.1: Kernel options for NFS

Securing the portmapper

First, make sure the portmapper has support for the tcp wrapper built-in (since it isn't started through inetd, portmapper needs its own built-in tcpwrapper support). You can test this by running **ldd /sbin/portmap** [2]. 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).

[2] The location of the portmapper may vary.

portmap and rpcbind

Some linux distributions use portmap. Other linux distributions use rpcbind.

The portmap daemon is replaced by rpcbind. Rpcbind has more features, like ipv6 support and nfs4 support.

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 Exporting NFS and 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 Section 9.2.6.

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. In 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 Exporting NFS.

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, the status daemon **rpc.statd** implements only a reboot notification service. It is a user-space daemon - even on systems with kernel-space 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 ❷hostname(❸options) 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 [3], 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 `client5` *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:

[3] Well, in most cases

```
# /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. Exportfs and fstab shows some of the functionality of **exportfs**.

Command	Description
exportfs -r	reexport all directories
exportfs -a	export or unexport all directories
exportfs -ua	de-activate the export list (unexport all)

Table 9.2: Overview of **exportfs**

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.

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 shows how **showmount** can be used.

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

Table 9.3: Overview of **showmount**

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 showmount lists the currently active export list:

```
# showmount --exports
Export list for nfsshop:
/ client2.exworks,client3.exworks,client4.exworks
/home client9.exworks
```

The information is more sparse compared to the output of **cat /proc/fs/nfs/exports** shown earlier.

Without options showmount will show names of hosts currently connected to the system:

```
# showmount
Hosts on nfsshop:
client9.exworks
```

With the --directories option 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** 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 Configuring 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 ❶remote:/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 Section 9.2.8.

nfsvers=*n* Specifies the NFS version used for the transport (see Section 9.2.8). 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 showmount, the **showmount `--exports`** command lists the current exports for a server system. This can be used as a quick indication of the health of the created NFS system. Nevertheless, there are more sophisticated ways of doing this.

The /proc/mounts file

To see which file systems are mounted check /proc/mounts. It will also show nfs mounted filesystems.

```
$ cat /proc/mounts
```

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 above, 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 provides an overview of relevant options for **nfsstat**.

	rpc	nfs	both
server	`-sr`	`-sn`	`-s`
client	`-cr`	`-cn`	`-c`
both	`-r`	`-n`	`-nr`

Table 9.4: Some options for the **nfsstat** program

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 completely 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 containing trusted users: 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 on (see Secure 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 wish to protect.

Firewall software The problem with tcp-wrapper support is that there already is a connection inside the host at the time that the connection-request is refused. If a security-related bug exists within either the tcp-wrapper library (not very likely) or the daemon that contains the support, unwanted access could be granted. Or worse. Firewall software (e.g., iptables) can make the kernel block connections before they enter the host. You may 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. How to set up a firewall is shown in detail in Chapter 12.

Preventing human error

Simple human error in combination with bad naming may also result in 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 with *read-write* permissions.

Mount read-only Mounting a remote filesystem *read-only* can prevent accidental erasure. So, mount *read-only* whenever 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 has been 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. At the time of this writing (May 2014) versions 4.0 and 4.1 have been released; version 4.2 is being developed. More about NFS version 4 and differences between earlier versions is covered in Section 9.2.8.

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 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 provides an overview of the most important files and software related to NFS.

program or file	description
The kernel	provides NFS support
The **portmap**per	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

Table 9.5: Overview of NFS-related programs and files

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 provides an overview.

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	new standard	kernel	tcp

Table 9.6: Overview of NFS protocol versions

The trends that can be seen in table Table 9.6 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. Also, `tcp` is a more reliable protocol by design, compared to `udp`. See Zadok01 for a discussion about this.

NFSv4

NFS version 4 (NFSv4) offers some new features compared to its predecessors. Instead of exporting multiple filesystems, NFSv4 exports a single pseudo file system for each client. The origin for this pseudo file system may be from different filesystems, but this remains transparent to the client.

NFSv4 offers an extended set of attributes, including support form MS Windows ACL's. Although NFSv4 offers enhanced security features compared to previous versions of NFS, and has been around since 2003, it was never widely adopted. Users are encouraged to implement NFSv4.1 which has been ratified in January 2010. See NFSv4.2 for detailed information about the different NFS versions and their characteristics.

Questions and answers

File Sharing

1. *By which means can the smb configuration be checked?*

   This can be done by using `testparm`. testparm

2. *What does root_squash do?*

   All requests made by the user *root* on the client will be executed as the user *nobody* on the server. root_squash [278]

3. *Why do you have to be sure that there is no space between the hostname and the user rights between braces in `/etc/exports` ?*

   As it gives all systems access with the user rights between the braces. space [278]

4. *In which file is samba configured?*

   Samba is configured via `/etc/samba/smbd.conf` Samba config file

5. *What does the mount option* soft *do?*

   The system will try to perform the mount until an RPC (portmapper) timeout occurs. soft

6. *What are security issues in NFS version 2?*

   It is possible to guess file handles and to sniff data. Securing NFS [285]

Chapter 10

Network Client Management (210)

This topic has a total weight of 11 points and contains the following 4 objectives:

Objective 210.1; DHCP Configuration (2 points) Candidates should be able to configure a DHCP server. This objective includes setting default and per client options, adding static hosts and BOOTP hosts. Also included is configuring a DHCP relay agent and maintaining the DHCP server.

Objective 210.2; PAM authentication (3 points) The candidate should be able to configure PAM to support authentication using various available methods.

Objective 210.3; LDAP client usage (2 points) Candidates should be able to perform queries and updates to an LDAP server. Also included is importing and adding items, as well as adding and managing users.

Objective 210.4; Configuring an OpenLDAP server (4 points) Candidates should be able to configure a basic OpenLDAP server including knowledge of LDIF format and essential access controls. An understanding of the role of SSSD in authentication and identity management is included.

DHCP Configuration (210.1)

Candidates should be able to configure a DHCP server. This objective includes setting default and per client options, adding static hosts and BOOTP hosts. Also included is configuring a DHCP relay agent and maintaining the DHCP server.

Key Knowledge Areas

DHCP configuration files, terms and utilities

Subnet and dynamically-allocated range setup

Terms and Utilities

- `dhcpd.conf`
- `dhcpd.leases`
- `/var/log/daemon.log` and `/var/log/messages`
- **arp**
- **dhcpd**
- **radvd**
- `radvd.conf`

What is DHCP?

The "Dynamic Host Configuration Protocol" (DHCP) is a protocol that allows computers to get configuration information about the network from the network. addresses are "leased" from servers to clients for a period of time. There is a separate protocol for assigning IPv6 addresses called DHCPv6, although the "Neighbour Discovery Protocol" (NDP) better fits this purpose.

The process of requesting and assigning addresses works as follows:

- When a computer starts, it sends a request to the network.
- Any DHCP servers that receive this request decide what address and other configuration options to assign to the client. This is typically based on things like: which network the request arrived on, or the MAC address of the interface that sent the request.
- Each server sends a packet which offers to assign the address to the client.
- The client decides which offer to accept, and sends a message to the server confirming the choice.
- The server acknowledges that it has recorded this address.

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.

Hosts using IPv6 are actually capable of assigning themselves link-local IP addresses using stateless autoconfiguration. DHCPv6 or NDP may be used to assign additional globally unique addresses and other configuration parameters. NDP is described in further detail in the section on **radvd**

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 behaviour 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 it 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.

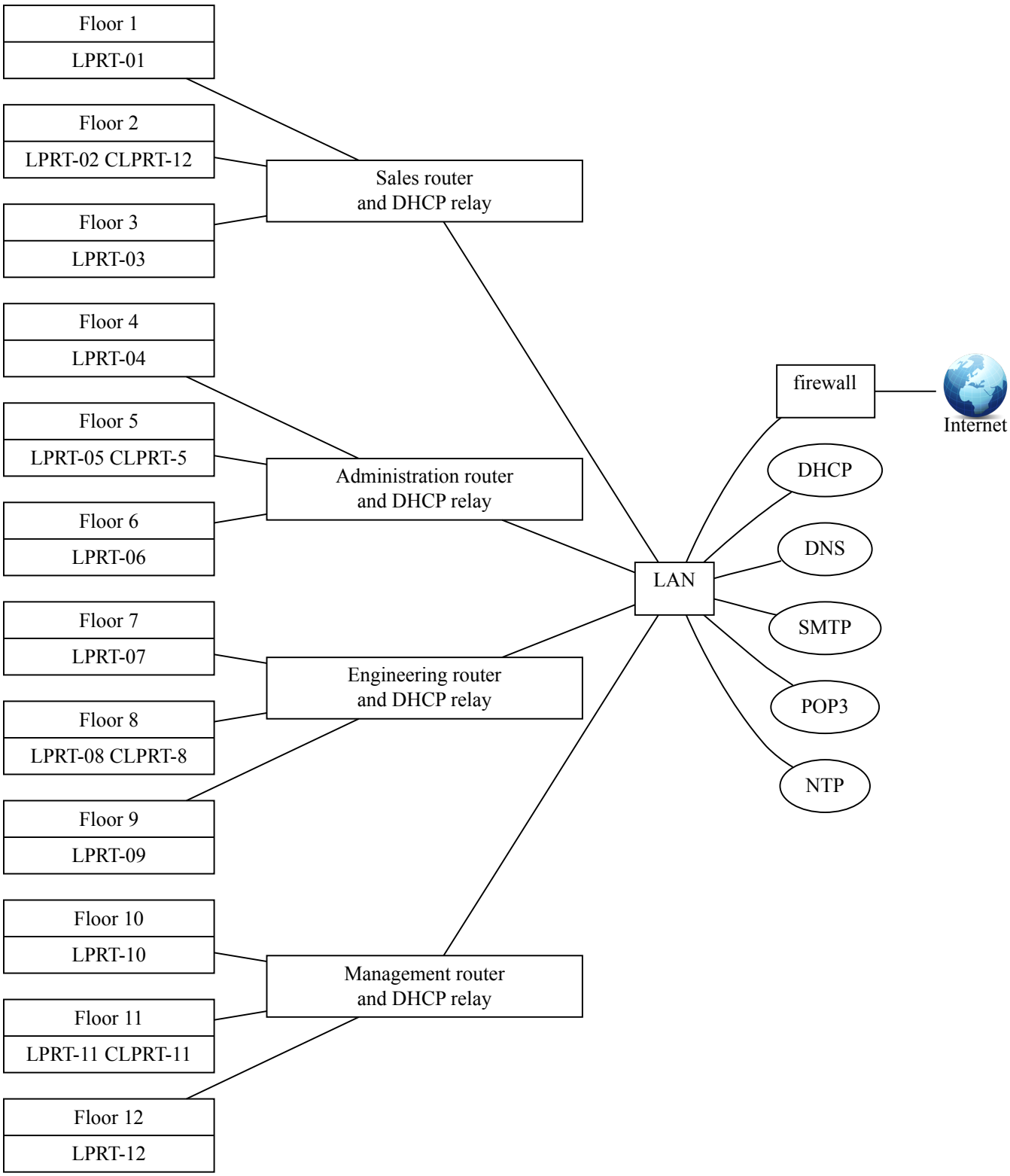
An example

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 colour 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:



Network layout of the company network.

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; in other words: the netmask used is /20 or 255.255.240.0), the subnets could be set up

as follows:

Dept.	Floor	IP range	Router	Description
0001	0001	21.31.17.0 - 21.31.17.255	21.31.17.1	Sales floor #1
0001	0010	21.31.18.0 - 21.31.18.255		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	21.31.36.1	Administration #4
0010	0101	21.31.37.0 - 21.31.37.255		Administration #5
0010	0110	21.31.38.0 - 21.31.38.255		Administration #6
0011	0111	21.31.55.0 - 21.31.55.255	21.31.55.1	Engineering floor #7
0011	1000	21.31.56.0 - 21.31.56.255		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	21.31.74.1	Management floor #10
0100	1011	21.31.75.0 - 21.31.75.255		Management floor #11
0100	1100	21.31.76.0 - 21.31.76.255		Management floor #12

Table 10.1: The first two octets are 21.31

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.

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

Table 10.2: Company-wide services

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.

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.

Department	Service	Description	IP-address	Name
Sales	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
	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
Administration	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
	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
Engineering	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
	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
Management	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
	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

Table 10.3: Subnet-dependent Services

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 instalment 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 reserved/used for the host address (as denoted by the subnet mask) 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 statement 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 21.31.37.0 netmask 255.255.240.0 {
    # Floor #5 specific parameters
    range 21.31.37.11 21.31.37.210;
  }
  subnet 21.31.38.0 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 21.31.56.0 netmask 255.255.240.0 {
    # Floor #8 specific parameters
    range 21.31.56.11 21.31.56.210;
  }
  subnet 21.31.57.0 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 21.31.75.0 netmask 255.255.240.0 {
    # Floor #11 specific parameters
    range 21.31.75.11 21.31.75.210;
  }
  subnet 21.31.76.0 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 behaviour

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. On the DHCP clients the leased IP addresses are kept in the file `dhclient.leases`.

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.

Logging

By default the DHCP server logs using syslogd, although many Linux distributions have exchanged syslogd for Systemd's journald. Logging is configured in the `dhcpd.conf` file using the log-facility keyword. This statement causes the DHCP server to do all of its logging on the specified log facility once the dhcpd.conf file has been read. By default the DHCP server logs to the daemon facility. Possible log facilities include auth, authpriv, cron, daemon, ftp, kern, lpr, mail, mark, news, ntp, security, syslog, user, uucp, and local0 through local7. Not all of these facilities are available on all systems, and there may be other facilities available on other systems. In addition to setting log-facility value, you may need to modify your syslog.conf file to configure logging of the DHCP server. For example, you might add a line like this:

```
  local7.debug /var/log/dhcpd.log
```

The syntax of the syslog.conf file may be different on some operating systems - consult the syslog.conf manual page to be sure. To get syslog to start logging to the new file, you must first create the file with correct ownership and permissions (usually, the same owner and permissions of your /var/log/messages or /usr/adm/messages file should be fine) and send a SIGHUP to syslogd. Note that **journald** does not log to a plaintext file; it uses a binairy format instead. To view messages specific for **dhcpd**, you have to filter these out using the **journalctl** command.

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* .

The DHCP Relay Agent listens for DHCP and BOOTP queries and responses. When a query is received from a client, dhcrelay forwards it to the list of DHCP servers specified on the command line. When a reply is received from a server, it is broadcast or unicast (according to the relay agent's ability or the client's request) on the network from which the original request came. If no interface names are specified on the command line dhcrelay will identify all network interfaces, elimininating non-broadcast interfaces if possible, and attempt to configure each interface.

Please consult the man page (**man dhcrelay**) for further details.

Assigning addresses in IPv6 networks

The previous paragraphs were mainly concerned with automatic IP assignment in IPv4 networks. Networks using IPv6 often use the "Neighbor Discovery Protocol" to obtain an IP address that is valid for the network. In Linux, this protocol is handled by the "Router Advertisement" (**radvd**) daemon.

The process for IPv6 address assignment works a bit different from IPv4 networks, because IPv6 hosts always assign link-local addresses for IPv6-enabled interfaces automatically, without any help from external hosts. NDP builds upon this *stateless autoconfiguration* process by distributing "prefixes" instead of addresses. Using the prefix (which is basically the network part of an IP address) obtained through NDP, hosts can assign valid IPv6 addresses to themselves. So in contrast with the DHCP daemon on IPv4 networks, NDP has no concept of IP pools and leases.

Instead of requesting an address, clients send "router sollicitation" requests to obtain a valid IPv6 prefix. The **radvd** daemon responds to these requests with router advertisement messages. These messages contain the routing prefix used on the link, the maximum transmission unit (MTU), and the address of the responsible default router.

The **radvd** daemon is configured by `/etc/radvd.conf`, which has to contain at least the interface the daemon should listen on and the prefix it has to serve. Additionally, **radvd** can periodically re-advertise its prefixes to hosts on the same network. If you wish, you can also configure the lifetime of the IPv6 addresses that hosts configure for themselves.

A typical `radvd.conf` would look similar to the following:

```
interface eth0 {
        AdvSendAdvert on;
        MinRtrAdvInterval 3;
        MaxRtrAdvInterval 10;
        prefix 2001:0db8:0100:f101::/64 {
                AdvOnLink on;
                AdvAutonomous on;
                AdvRouterAddr on;
        };
};
```

PAM authentication (210.2)

The candidate should be able to configure PAM to support authentication using various available methods.

Key Knowledge Areas

PAM configuration files, terms and utilities

passwd and shadow passwords

basic SSSD functionality for LDAP authentication

Terms and Utilities

- `/etc/pam.d`

- `pam.conf`
- `nsswitch.conf`
- `pam_unix, pam_cracklib, pam_limits, pam_listfile`

Resources

Resources: dracut;

What is PAM?

PAM is the acronym 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, unless 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.

Modules

pam_unix

This module configures authentication via `/etc/passwd` and `/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
```

pam_nis

This module configures 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
```

pam_ldap

This module configures 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
```

pam_cracklib

This plugin provides strength-checking for passwords. This is done by performing a number of checks to ensure passwords are not too weak. It checks the password against dictonaries, the previous password(s) and rules about the use of numbers, upper and lowercase and other characters.

```
#%PAM-1.0
#
# These lines allow a md5 systems to support passwords of at least 14
# bytes with extra credit of 2 for digits and 2 for others the new
# password must have at least three bytes that are not present in the
# old password
#
password  required pam_cracklib.so \
difok=3 minlen=15 dcredit= 2 ocredit=2
password  required pam_unix.so use_authtok nullok md5
```

pam_limits

The pam_limits PAM module sets limits on the system resources that can be obtained in a user-session. Users of uid=0 are affected by this limits, too. By default limits are taken from the `/etc/security/limits.conf` config file. Then individual files from the `/etc/security/limits.d/` directory are read. The files are parsed one after another in the order of "C" locale. The effect of the individual files is the same as if all the files were concatenated together in the order of parsing. If a config file is explicitely specified with a module option then the files in the above directory are not parsed. The module must not be called by a multithreaded application.

pam_listfile

This module allows or denies an action based on the presence of the item in a listfile. A listfile is a textfile containing a list of usernames, one username per line. The type of item can be set via the configuration parameter item and can have the value of user, tty, rhost, ruser, group, or shell. The *sense* configuration parameter determines whether the entries in the list are allowed. Possible values are allow and deny.

SSSD

Configure SSSD for LDAP authentication

The following steps describe the configuration of SSSD to use LDAP for authentication:

1. The following packages need to be installed:

```
sssd-client
sssd-common
sssd-common-pac
sssd-ldap
sssd-proxy
python-sssdconfig
authconfig
authconfig-gtk
```

Use your package manager to install these packages.

2. Check the current settings for sssd, if any:

```
# authconfig test
```

This will show you the current settings which are already in place. Also check for an existing `/etc/sssd/sssd.conf` file. On a fresh installation you can expect all settings to be disabled and that the sssd.conf file will not be present.

3. Now configure sssd:

```
# authconfig \
--enablesssd \
--enablesssdauth \
--enablelocauthorize \
--enableldap \
--enableldapauth \
--ldapserver=ldap://ldap.example.com:389 \
--disableldaptls \
--ldapbasedn=dc=example,dc=com \
--enablerfc2307bis \
--enablemkhomedir \
--enablecachecreds \
--update
```

4. Check the configuration in `/etc/sssd/sssd.conf`.

In case you're using TLS make sure that the `ldap_tls_cacertdir` and `ldap_tls_cacert` parameters are configured correctly and point to your certificates. Also change `ldap_id_use_start_tls` to "True".

To effect the changes, run:

```
# systemctl restart sssd
```

Verify that all changes are effective by running:

```
# authconfig test
```

5. Update `/etc/openldap.conf` to use the same ldap settings. Your `ldap.conf` file will look like this:

```
SASL_NOCANON on
URI ldaps://ldap.example.com:389
BASE dc=example,dc=com
TLS_REQUIRE never
TLS_CACERTDIR /etc/pki/tls/cacerts
TLS_CACERT /etc/pki/tls/certs/mybundle.pem
```

Please note that `TLS_REQUIRE` is set to never. This is done in order to avoid issues with application stacks like `PHP`, which have difficulties with `LDAPS` and `TLS`.

6. Make sure that sssd is up and running and that it will be started after a system reboot. Run **systemctl** `status sssd` to check this. To start sssd, run **systemctl** `start sssd` and to make sssd persistent across reboots, run **systemctl** `enable sssd`.

LDAP client usage (210.3)

Candidates should be able to perform queries and updates to an LDAP server. Also included is importing and adding items, as well as adding and managing users.

Key Knowledge Areas

LDAP utilities for data management and queries

Change user passwords

Querying the LDAP directory

Terms and Utilities

- **ldapsearch**
- **ldappasswd**
- **ldapadd**
- **ldapdelete**

LDAP

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.

LDAP is a client/server system. The server can use a variety of databases to store a directory, each optimized for quick and copious read operations. When an LDAP client application connects to an LDAP server, it can either query a directory or attempt to modify it. In the event of a query, the server either answers the query locally, or it can refer the querent to an LDAP server which does have the answer. If the client application is attempting to modify information within an LDAP directory, the server verifies that the user has permission to make the change and then adds or updates the information.

LDAP TERMINOLOGY

entry A single unit within an LDAP directory. Each entry is identified by its unique *Distinguished Name* (DN).

attributes Information directly associated with an entry. For example, an organization could be represented as an LDAP entry. Attributes associated with the organization might be its fax number, its address, and so on. People can also be represented as entries in the LDAP directory. Common attributes for people include the person's telephone number and email address.

Some attributes are required, while other attributes are optional. An objectclass definition sets which attributes are required and which are not for each entry. Objectclass definitions are found in various schema files, located in the `/etc/openldap/schema/` directory.

LDIF The LDAP Data Interchange Format (LDIF) is an ASCII text representation of LDAP entries. Files used for importing data to LDAP servers must be in LDIF format. An LDIF entry looks similar to the following example:

```
[<id>]
dn: <distinguished name>
<attrtype>: <attrvalue>
<attrtype>: <attrvalue>
<attrtype>: <attrvalue>
```

Each entry can contain as many <attrtype>: <attrvalue> pairs as needed. A blank line indicates the end of an entry.

> **Note**
> All <attrtype> and <attrvalue> pairs must be defined in a corresponding schema file to use this information.

Any value enclosed within a "<" and a ">" is a variable and can be set whenever a new LDAP entry is created. This rule does not apply, however, to <id>. The <id> is a number determined by the application used to edit the entry.

ldapsearch

ldapsearch is a shell-accessible interface to the ldap_search(3) library call. **ldapsearch** opens a connection to an LDAP server, binds, and performs a search using specified parameters. The filter should conform to the string representation for search filters as defined in *RFC 2254*.

LDAP Filters

Equality	=	Creates a filter which requires a field to have a given value. For example, cn=Eric Johnson.
Presence	=*	Wildcard to represent that a field can equal anything except NULL. So it will return entries with one or more values. For example, cn=* manager=*
Substring	=string* string	Returns entries containing attributes containing the specified substring. For example, cn=Bob* cn=*John* cn=E*John. The asterisk (*) indicates zero (0) or more characters.
Approximate	~=	Returns entries containing the specified attribute with a value that is approximately equal to the value specified in the search filter. For example, cn~=suret l~=san franciso could return cn=sarette l=san francisco
Greater than or equal to	>=	Returns entries containing attributes that are greater than or equal to the specified value.
Less than or equal to	<=	Returns entries containing attributes that are less than or equal to the specified value.
Parentheses	()	Separates filters to allow other logical operators to function.
And	&	Boolean operator. Joins filters together. All conditions in the series must be true. For example, (&(filter)(filter)...).
Or	|	Boolean operator. Joins filters together. At least one condition in the series must be true. For example, (|(filter)(filter)...).
Not	!	Boolean operator. Excludes all objects that match the filter. Only one filter is affected by the NOT operator. For example, (!(filter))

Table 10.4: LDAP field operators

Boolean expressions are evaluated in the following order:

- Innermost to outermost parenthical expressions first.
- All expressions from left to right.

Examples:

```
ldapsearch -h myhost -p 389 -s base -b "ou=people,dc=example,dc=com" "objectclass=*"
```

This command searches the directory server myhost, located at port 389. The scope of the search (-s) is base, and the part of the directory searched is the base DN (-b) designated. The search filter "objectclass=*" means that values for all of the entry's object classes are returned. No attributes are returned because they have not been requested. The example assumes anonymous authentication because authentication options are not specified.

```
ldapsearch -x "(|(cn=marie)(!(telephoneNumber=9*)))"
```

This example shows how to search for entries that have a cn of marie OR do NOT have a telephoneNumber beginning with 9.

ldappasswd

ldappasswd - change the password of an LDAP entry

ldappasswd is a tool to set the password of an LDAP user. ldappasswd uses the LDAPv3 Password Modify (*RFC 3062*) extended operation.

ldappasswd sets the password associated with the user (or an optionally specified user). If the new password is not specified on the command line and the user doesn't enable prompting, the server will be requested to generate a password for the user.

Example:

```
ldappasswd -x -h localhost -D "cn=root,dc=example,dc=com" \
  -s secretpassword -W uid=admin,ou=users,ou=horde,dc=example,dc=com
```

Set the password for "uid=admin,ou=users,ou=horde,dc=example,dc=com on localhost".

ldapadd

ldapadd - LDAP add entry tool

ldapadd is implemented as a link to the **ldapmodify** tool. When invoked as **ldapadd** the `-a` (add new entry) flag is turned on automatically.

Option: **ldapmodify** `-a`

`-a` Adds new entries. The default for **ldapmodify** is to modify existing entries. If invoked as **ldapadd**, this option is always set.

Example:

```
ldapadd -h myhost -p 389 -D "cn=orcladmin" -w welcome -f jhay.ldif
```

Using this command, user orcladmin authenticates to the directory myhost, located at port 389. The command then opens the file jhay.ldif and adds its contents to the directory. The file might, for example, add the entry "uid=jhay,cn=Human Resources,cn=example,dc=com" and its object classes and attributes.

ldapdelete

ldapdelete - LDAP delete entry tool

ldapdelete is a shell-accessible interface to the ldap_delete_ext(3) library call.

ldapdelete opens a connection to an LDAP server, binds, and deletes one or more entries. If one or more DN arguments are provided, entries with those Distinguished Names are deleted.

Example:

```
ldapdelete -h myhost -p 389 -D "cn=orcladmin" -w welcome \
"uid=hricard,ou=sales,ou=people,dc=example,dc=com"
```

This command authenticates user orcladmin to the directory myhost, using the password welcome. Then it deletes the entry "uid=hricard,ou=sales,ou=people,dc=example,dc=com".

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*
- *OpenLDAP Software 2.4 Administrator's Guide*
- *The LDAP Linux HOWTO*
- *The Internet FAQ archives* where RFC's and other documentation can be searched and found.

Configuring an OpenLDAP server (210.4)

Candidates should be able to configure a basic OpenLDAP server including knowledge of LDIF format and essential access controls.

Key Knowledge Areas

OpenLDAP

Directory based configuration

Access Control

Distinguished Names

Changetype Operations

Schemas and Whitepages

Directories

Object IDs, Attributes and Classes

Terms and Utilities

- **slapd**
- `slapd.conf`
- LDIF
- **slapadd**
- **slapcat**
- **slapindex**
- `/var/lib/ldap/`
- **loglevel**

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

OpenLDAP

OpenLDAP uses **slapd** which is the stand-alone LDAP daemon. It listens for LDAP connections on any number of ports (389 by default), responding to the LDAP operations it receives over these connections. OpenLDAP is typically started at boot time.

It can be installed either from source obtaining it from *OpenLDAP software* but most linux distributions deliver it through their packagemanagement system like yum or apt.

In Openldap, directory entries are arranged in a hierarchical tree-like structure. Traditionally, this structure reflected the geographic and/or organizational boundaries. Entries representing countries appear at the top of the tree. Below them are entries representing states and national organizations. Below them might be entries representing organizational units, people, printers, documents, or just about anything else.

Access Control

While the OpenLDAP directory gets filled the data becomes more critical. Some data might be protected by law or be confidential in an other way. Therefore access to the directory needs to be controlled. The default policy allows read access to all clients. Regardless of what access control policy is defined, the olcRootDN is always allowed full rights (i.e. auth, search, compare, read, and write) on everything and anything.

Access to **slapd** entries and attributes is controlled by the **olcAccess** attribute, whose values are a sequence of access rules. They begin with the access directive followed by a list of conditions:

```
olcAccess: to <what>
     by <who> <type of access>
     by <who> <type of access>
```

For example:

```
olcAccess: to attrs=userPassword
     by anonymous auth
     by self write
     by * none
```

This access specification is used to keep a user's password protected. It allows anonymous users an authentication comparison on a password for the purpose of of logging on. Additionally it grants a user permission to change his password. The bottom line denies everyone else any access to the password.

Alternatively we could grant users permission to update all their data with access speficifations like the following:

```
olcAccess: to attrs=userPassword
     by anonymous auth
     by * none
olcAccess: to *
             by self write
             by * none
```

Distinguished Names

A distinguished name (DN) is the name (set of attributes) which uniquely identifies an entry in the OpenLDAP directory and corresponds to the **path** which has to be traversed to reach that entry. A DN contains an attribute and value pair separated by commas.

For example:

```
cn=John Doe,ou=editing,o=Paris,c=F
cn=Jane Doe,ou=editing,o=London,c=UK
cn=Tom Jones,ou=reporting,o=Amsterdam,c=NL
```

Any of the attributes defined in the directory schema may be used to make up a DN. The order of the component attribute value pairs is important. The DN contains one component for each level of the directory hierarchy from the root down to the level where the entry resides. LDAP DNs begin with the most specific attribute and continue with progressively broader attributes. The first component of the DN is referred to as the Relative Distinguished Name (RDN). It identifies an entry distinctly from any other entries that have the same parent.

An example to create an entry for a person:

```
dn: cn=John Doe,o=bmi,c=us
objectclass: top
objectclass: person
cn: John Doe
sn: Doe
telephonenumber: 555-111-5555
```

Some characters have special meaning in a DN. For example, = (equals) separates an attribute name and value and comma separates attribute=value pairs. The special characters are: comma, equals, plus,less than, greater than, number sign, semicolon, backslash, quotation mark.

A special character can be escaped in an attribute value to remove the special meaning. To escape these special characters or other characters in an attribute value in a DN string, use the following methods:

If a character to be escaped is one of the special characters, precede it by a backslash ("\" ASCII 92). This example shows a method of escaping a comma in an organization name:

```
CN=L. Eagle,O=Sue\, Grabbit and Runn,C=GB
```

slapd-config

OpenLDAP 2.3 and later have transitioned to using a dynamic runtime configuration engine, **slapd-config**. The older style `slapd.conf` file is still supported, but its use is deprecated and support for it will be withdrawn in a future OpenLDAP release.

> **Note**
> Although the slapd-config system stores its configuration as (text-based) LDIF files, you should never edit any of the LDIF files directly. Configuration changes should be performed via LDAP operations, e.g. **ldapadd**, **ldapdelete**, or **ldapmodify**.

Depending on the linux distribution the **slapd-config** configuration tree `slapd.d` may be located in `/etc/openldap` or `/usr/local/etc/openldap`.

An example might look like this:

```
/etc/openldap/slapd.d
|-- cn=config
|~~ |-- cn=module{0}.ldif
|~~ |-- cn=schema
|~~ |~~ |-- cn={0}core.ldif
|~~ |~~ |-- cn={1}cosine.ldif
|~~ |~~ `-- cn={2}inetorgperson.ldif
|~~ |-- cn=schema.ldif
|~~ |-- olcDatabase={0}config.ldif
|~~ |-- olcDatabase={-1}frontend.ldif
|~~ `-- olcDatabase={1}hdb.ldif
`-- cn=config.ldif
```

The `slapd.d` tree has a very specific structure. The root of the tree is named **cn=config** and contains global configuration settings. Additional settings are contained in separate child entries.

These may be the following:

- Dynamically loaded modules in the `cn=module{0}.ldif`
- Schema definitations in the `cn=schema` directory (more about the topic of schema's will follow below)
- Backend-specific configuration in the `cn=Database={1}hdb.ldif`
- Database-specific configuration in the `cn=Database={0}config.ldif`

The general layout of the LDIF (for more information on LDIF refer to the section below) that is used to create the configuration tree is as follows:

```
# global configuration settings
dn: cn=config
objectClass: olcGlobal
cn: config
<global config settings>

# schema definitions
dn: cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema
<system schema>

dn: cn={X}core,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: {X}core
<core schema>

# additional user-specified schema
...

# backend definitions
dn: olcBackend=<typeA>,cn=config
objectClass: olcBackendConfig
olcBackend: <typeA>
<backend-specific settings>

# database definitions
dn: olcDatabase={X}<typeA>,cn=config
objectClass: olcDatabaseConfig
olcDatabase: {X}<typeA>
<database-specific settings>

# subsequent definitions and settings
...
```

For the domain `example.com` the configuration file might look like this:

```
dn: olcDatabase=hdb,cn=config
objectClass: olcDatabaseConfig
objectClass: olcHdbConfig
olcDatabase: hdb
olcSuffix: dc=example,dc=com
olcRootDN: cn=Manager,dc=example,dc=com
olcRootPW: secret
olcDbDirectory: /var/lib/ldap
```

It is more secure to generate a password hash using **slappasswd** instead of the plain text password secret as in the example above. In that case the **olcRootPW** line would be changed into something like the following:

```
olcRootPW: {SSHA}xEleXlHqbSyi2FkmObnQ5m4fReBrjwGb
```

The **olcLogLevel** directive specifies at which debugging level statements and operation statistics should be syslogged. Log levels may be specified as integers or by keyword. Multiple log levels may be used and the levels are additive.

Available levels are:

```
1      (0x1 trace) trace function calls
2      (0x2 packets) debug packet handling
4      (0x4 args) heavy trace debugging (function args)
8      (0x8 conns) connection management
16     (0x10 BER) print out packets sent and received
32     (0x20 filter) search filter processing
64     (0x40 config) configuration file processing
128    (0x80 ACL) access control list processing
256    (0x100 stats) stats log connections/operations/results
512    (0x200 stats2) stats log entries sent
1024   (0x400 shell) print communication with shell backends
2048   (0x800 parse) entry parsing
16384  (0x4000 sync) LDAPSync replication
32768  (0x8000 none) only messages that get logged whatever log level is set
```

For example:

```
olcLogLevel: -1
```

This will cause lots and lots of debugging information to be logged.

```
olcLogLevel: conns filter
```

This will only log the connection and search filter processing.

```
olcLogLevel: stats
```

Basic stats logging is configured by default. However, if no olcLogLevel is defined, no logging occurs (equivalent to a 0 level).

Note that the actual OpenLDAP database holding the user data is not located in the `slapd.d` configuration directory tree. Its location may be changed with the olcDbDirectory directory (see the example above) but by convention it is usually `/var/lib/ldap`.

Its contents typically looks like this:

```
$ ls -l /var/lib/ldap
total 1168
-rw-r--r--. 1 ldap ldap    4096 Dec 12 14:29 alock
-rw-------. 1 ldap ldap    8192 Dec  2 21:31 cn.bdb
-rw-------. 1 ldap ldap 2351104 Dec 12 14:29 __db.001
-rw-------. 1 ldap ldap  819200 Dec 12 14:29 __db.002
-rw-------. 1 ldap ldap  163840 Dec 12 14:29 __db.003
-rw-rw-r--. 1 ldap ldap     104 Dec  2 21:12 DB_CONFIG
-rw-------. 1 ldap ldap    8192 Dec  2 21:31 dn2id.bdb
-rw-------. 1 ldap ldap   32768 Dec  2 21:31 id2entry.bdb
-rw-------. 1 ldap ldap    8192 Dec  2 21:31 objectClass.bdb
```

LDIF

All modifications to the OpenLDAP database are formatted in the LDIF format. LDIF stands for LDAP Data Interchange Format. It is used by OpenLDAP's tools like `slapadd` in order to add data to the database. An example of a LDIF file:

```
cat adduser.ldif

# John Doe's Entry
dn: cn=John Doe,dc=example,dc=com
```

```
cn: John Doe
cn: Johnnie Doe
objectClass: person
sn: Doe
```

Multiple entries are separated using a blank line. **Slapcat** can be used to export information from the LDAP database in the LDIF format.

For example:

```
slapcat -l all.ldif
```

This will generate a file called `all.ldif` which contains a full dump of the LDAP database.

The generated output can be used by **slapadd** to import the data into an LDAP database.

For example:

```
slapadd -l all.ldif
```

Sometimes it may be necessary to regenerate LDAP's database indexes. This can be done using the `slapindex` tool. It may also be used to regenerate the index for a specific attribute like the UID:

```
slapindex uid
```

Note that **slapd** should not be running (at least, not in read-write mode) when the command is run to ensure consistency of the database.

Directories

A directory can be contrived of as an hierarchically organized collection of data. The best known example probably is the telephone directory, but the file system directory is another one. Generally speaking a directory is a database that is optimized for reading, browsing and searching. OpenLDAP directories contain descriptive, attribute-based information. They do not support the roll-back mechanisms or complicated transactions that are found in Relational Data Base Management Systems (RDBMS's). Updates are typically simple all-or-nothing changes, if allowed at all. This type of directories are designed to give quick responses to high-volume lookup or search operations. OpenLDAP directories can be replicated to increase availability and reliability. Replicated databases can be temporarily out-of-sync but will be synchronized eventually.

Schemas and Whitepages

Schemas are the standard way of describing the structure of objects that may be stored inside a directory. A whitepages schema is a data model for organizing the data contained in entries in a directory service such as an address book or LDAP. In a whitepages directory, each entry typically represents an individual that makes use of network resources, such as by receiving email of having an account to log in to a system. LDAP schemas are used to formally define attributes, object classes, and various rules for structuring the directory information tree. Usually schemas are configured in `slapd-config` LDIF using the include directive.

For example:

```
include: file:///etc/openldap/schema/core.ldif
include: file:///etc/openldap/schema/cosine.ldif
include: file:///etc/openldap/schema/inetorgperson.ldif
```

The first line imports the core schema, which contains the schemas of attributes and object classes necessary for standard LDAP use. The cosine.schema imports a number of commonly used object classes and attributes, including those used for storing document information and DNS records. The third provides the inetOrgPerson object class definition and its associated attribute definitions. Other schemas are available with OpenLDAP (in `/etc/openldap/schema`); refer to the OpenLDAP Software 2.4 Administrator's guide for more information.

References

- *OpenLDAP Software 2.4 Administrator's Guide*
- *Directory Service*
- *Lightweight Directory Access Protocol*
- *System Security Services Daemon*
- *White Pages Schema*

Questions and answers

Network Client Management

1. *What does LDAP stand for?*

   LDAP stands for Lightweight Directory Access Protocol LDAP [305]

2. *What is a **dn**?*

   The *distinguishedName*, or **dn**, consists of the set of attributes which uniquely identify an LDAP entry. This set of attributes corresponds to the **path** which has to be traversed to reach that entry. DN [307]

3. *What is a shared-network declaration?*

   A shared-network declaration is used if there are multiple subnets on the same physical network. shared-network [289]

4. *In which file does the DHCP server keep a database of the leases?*

   `/var/dhcp/dhcpd.leases` DHCP lease [298]

5. *How would you reload the DHCP server?*

   `/etc/init.d/dhcp restart` DHCP restart [298]

6. *What is a static host?*

   A static host is a host that always gets the same IP-address from the DHCP-server. Static host [296]

7. *What is a lease?*

   A lease is the amount of time a client may use the IP address it has received from the DHCP-server. lease [297]

8. *What does the acronym DHCP stand for?*

   DHCP stands for Dynamic Host Configuration Protocol. DHCP [289]

9. *What is the function of the type auth?*

   The type **auth** checks the user's password against the password database(s). auth [301]

Chapter 11

E-Mail services (211)

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

Objective 211.1; Using e-mail servers (4 points) Candidates should be able to manage an e-mail server, including the configuration of e-mail aliases, e-mail quotas and virtual e-mail domains. This objective includes configuring internal e-mail relays and monitoring e-mail servers.

Objective 211.2; Managing local e-mail delivery (2 points) Candidates should be able to implement client e-mail management software to filter, sort and monitor incoming user e-mail.

Objective 211.3 Managing remote e-mail delivery (2 points) Candidates should be able to install and configure POP and IMAP daemons.

Using e-mail servers (211.1)

Candidates should be able to manage an e-mail server, including the configuration of e-mail aliases, e-mail quotas and virtual e-mail domains. This objective includes configuring internal e-mail relays and monitoring e-mail servers.

Key Knowledge Areas

Configuration files for postfix

Basic knowledge of the SMTP protocol

Awareness of sendmail and exim

Terms and Utilities

- Configuration files and commands for postfix
- `/etc/aliases`
- `/etc/postfix/`
- `/var/spool/postfix/`
- sendmail emulation layer commands
- mail-related logs in /var/log

Resources: PostfixTLSreadme, PostfixTFSreadme, SMTPauthHowto, RFC2487, raymii.org, PostfixGuide, the **man** pages for the various commands.

Basic knowledge of the SMTP protocol

Since October 2008, RFC 5321 is used to describe the .

When an has a message to transmit, it establishes a two-way transmission channel to an . The responsibility of the client is to transfer the message to one or more SMTP servers, or to report a failure to do so.

The means by which an message is presented to an SMTP client, and how the domain name to transfer the message to is determined, is a local matter that will not be addressed here.

Detailed information can be found at:*RFC5321*.

For demonstration and testing purposes a plain text SMTP session can be initiated with **telnet**, **nc** or **ncat**:

```
telnet mx1.unix.nl 25
Connected to mx1.unix.nl (213.154.248.146).
Escape character is '^]'.
220 mx1.unix.nl ESMTP Exim 4.63 Thu, 12 Aug 2010 08:50:30 +0200
ehlo snow.nl
250-mx1.unix.nl Hello mx1.unix.nl [81.23.226.83]
250-SIZE 52428800
250-PIPELINING
250-AUTH PLAIN LOGIN
250-STARTTLS
250 HELP
```

When using **nc** or **ncat** when performing the commands above, an additional **Enter** keystroke may be necessary. This is due to the differences in return characters that may exist between **telnet** and other, similar tools.

With **telnet** ***<smtp servername> 25*** initial contact with the SMTP server is established. In the example above mx1.unix.nl was used as the server. The server responds with the maximum message size, the pipelining, and the authentication capabilities of the server, in this case *AUTH PLAIN LOGIN*. The server is also able to use *STARTTLS*, and *HELP*.

- SIZE: when the size of the message the client wishes to send exceeds the set limit, the SMTP transaction will be aborted with an ERROR code pipelining. Also, when enabled, an SMTP client can transmit a group of SMTP commands (e.g., RSET, MAIL FROM:, RCPT TO:) without waiting for a response from the server.
- AUTH PLAIN LOGIN: the SMTP server is capable of handling a plain password/username authentication. This can be handy for mobile devices using the SMTP server while originating from different IP addresses.
- STARTTLS: The SMTP protocol itself doesn't include any form of encryption. With STARTTLS the communciation can be encrypted using certificates. This is fully described in RFC 3207.

Since we have established the initial connection we can now proceed:

```
mail from: nonexistent@snow.nl
250 OK
rcpt to: nonexistent@unix.nl
250 Accepted
```

The server responds with "250 OK" but when, for example, the "MAIL FROM:" command is followed by a misformatted or incomplete email address the server responds with "501: sender address must contain domain" or a similar error. With "RCPT TO: <email address>" the destination of the message is given. If the recipient's address is accepted the server responds with: "250 Accepted".

Now the knows who we are and to whom we wish to transmit a message. The also knows what the SMTP server is capable of. We then proceed with the transmission:

```
data
354 Enter message, ending with "." on a line by itself
mail from: Aiu <nonexistent@snow.nl>
From: <nonexistent@snow.nl>
To: Info <nonexistent@unix.nl>
```

```
Subject: Demo messages
Date: 12-02-2010 08:53:00
MesssageID: 31231233@snow.nl
This is a demonstration mesage.
.
250 OK id=1OjReS-0005kT-Jj
quit
```

With the *DATA* command we proceed with the contents of the message. The server responds with: “354 Enter message, ending with "." on a line by itself”.

Then the message contents is entered, starting with the message headers. These headers are used by the email client. In the example the commands *Mail From:*, *To:*, *Subject:*, *Date:* and *MessageID:* are used. *MessageID:* is a unique identifier generated by the SMTP client. These headers are required as described in RFC 5322.

After the headers have been entered, a blank line indicates that the actual text of the message commences. The message ends, as the *DATA* command response has indicated, with a “.” (without the quotes) on a line by itself. The server responds with: “250 OK id=1OjReS-0005kT-Jj”. The id is a unique SMTP server identifier and can be used for troubleshooting.

Note
To fight , some SMTP servers can check whether a message is RFC 5322 compliant. Regular SMTP clients are RFC 5322 compliant, but spammers often use not so regular SMTP clients and hence could send malformed messages.

Sendmail

Basic sendmail configuration steps are:

1. Download Sendmail
2. Set up Sendmail
3. Configure Sendmail
4. Build the Sendmail User Table
5. Add your domain names to Sendmail
6. Test your configuration file

Basic configuration of sendmail is described at: *Sendmail basic installation*.

An installation and operation guide for sendmail 8.12 can be found at: *Sendmail installation and operation guide*.

Sendmail can either be built from source or installed as sendmail binaries. With sendmail you must build a run-time configuration file. This is a file that sendmail reads when it starts which describes the mailers it knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. Although it can be quite complex, it can usually be built using an m4-based configuration language (is a general-purpose macro processor). Assuming you have the standard sendmail distribution, see cf/README for more information.

The sendmail configuration files are processed by m4 to facilitate local customization; the directory cf in the distribution directory contains the source files. It contains several subdirectories:

cf

Both site-dependent and site-independent descriptions of hosts. The files can be named after hosts (e.g., ucbvax.mc) when these are gateways, or after categories (such as generic-solaris2.mc) as a general description of an SMTP-connected host running Solaris 2.x. Files ending with .mc (“M4 Configuration”) are input definitions; the output is in the corresponding .cf file. The files’ general structure is described below.

domain

Site-dependent subdomain descriptions. These are tied to the way your organization wants to do addressing. For example, `domain/CS.Berkeley.EDU.m4` is our description for hosts in the CS.Berkeley.EDU subdomain. These are referenced using the DOMAIN m4 macro in the .mc file.

feature

Definitions of specific features that a particular host at your site might need. These are referenced using the *FEATURE* m4 macro. An example is use_cw_file that tells sendmail to read an `/etc/mail/local-host-names` file on startup to determine the set of local hosts.

hack

Local hacks, referenced using the *HACK* m4 macro. Try to avoid these. The point of having them here is to make it clear that they smell.

m4

Site-independent m4(1) include files that have information common to all configuration files. This can be thought of as an *#include* directory.

mailer

Definitions of mailers, referenced using the *MAILER* m4 macro. The types that are known in this distribution are fax, local, smtp, uucp, and usenet. For example, to include support for UUCP-based mailers, use *MAILER(uucp)*.

ostype

Definitions describing various operating system environments (such as the location of support files). These are referenced using the *OSTYPE* m4 macro.

sh

Shell files used by the m4 build process. You probably don't have to change these.

siteconfig

Local UUCP connectivity information. This directory has been replaced by the mailertable feature; any new configurations should use that feature to do UUCP (and other) routing. *The use of the siteconfig directory is deprecated.*

Make sure the m4 utility is available at the server. With it the .mc macro files can be converted to a sendmail.cf file. The recommended method to configure sendmail is to edit the macro files and not the sendmail.cf file itself.

Details of sendmail installation files can be seen at: *Sendmail installation and operation guide*.

Important sendmail files

`/etc/mail/sendmail.cf` - Main sendmail configuration file.

`/etc/mail/access` - The sendmail access database file can be created to accept or reject mail from selected domains, systems and users. Since it is a database, after editing the text file you have to use **makemap** to create or update it.

To create a new access map:

```
# makemap hash /etc/mail/access.db < /etc/mail/access
```

In the access file several actions may be defined:

OK Accept mail even if other rules in the running ruleset would reject it, for example, if the domain name is unresolvable. "Accept" does not mean "relay", but at most acceptance for local recipients. Thus, *OK* allows less than *RELAY*.

RELAY Accept mail addressed to the indicated domain or received from the indicated domain for relaying through your SMTP server. *RELAY* also serves as an implicit *OK* for the other checks.

REJECT Reject the sender or recipient with a general purpose message.

DISCARD Discard the message completely using the $#discard mailer. If it is used in check_compat, it affects only the designated recipient, not the whole message as it does in all other cases. This should only be used if really necessary.

SKIP This can only be used for host/domain names and IP addresses/nets. It will abort the current search for this entry without accepting or rejecting it but causing the default action.

`/etc/mail/local-host-names` - Local host names can be defined in the local-host-names file. Add each domain that should be considered local into /etc/mail/local-host-names.

`/etc/mail/virtusertable` - Used to map incoming email to a local account. With virtusertable, messages sent to one account can be distributed to two users. Also a "catch all" email address can be set up to route all mistyped email to one particular user to create a new virtusertable.

To generate the virtusertable database map:

```
# makemap hash /etc/mail/virtusertable < sourcefile
```

`/etc/mail/genericstable` - Used for outbound mail. Can be used to rewrite local usernames so they appear to have originated from a different host or domain.

`/etc/mail/genericsdomain` - Populate this file with the **hostname --long** command.

```
# hostname --long > genericsdomain
```

`/etc/mail/mailertable` - Used to route email from remote systems.

`/etc/mail/domaintable` - Can be used to transition from an old domain name to a new one.

`/etc/mail/aliases` - Used to redirect mail for local recepients. Each line of `/etc/aliases` has the format of *alias: user*. Two system aliases must always be present: *mailer_daemon: postmaster* and *postmaster: root*. You can use aliases for all kind of daemons, for example use *ntp: root*. Now you can add a line to redirect all mail to *root* to a specific user or group of administrators, for example *root: marc*. **newaliases** needs to be run after any change to this file.

```
# newaliases
```

If any update is made to one of the configuration files, sendmail needs to reload its' configuration:

```
# killall -HUP sendmail
```

Antirelaying

Starting with version 8.9, sendmail does not relay by default. When using an older sendmail version, make changes to the `sendmail.cf` or `access` files to make sure that it does not relay. Antirelaying tips are described at: *Controlling SMTP relaying tips - Sendmail.org*.

Sendmail test option

Sendmail can be run in test mode. Use the `-b` and `-t` options to do this. You need to run this as root.

```
# sendmail -bt
```

Sendmail and DNS

Make sure that the MX records for the MTA's are available in DNS. This can be checked with:

```
$ dig MX somedomain.com
```

Manual entries in sendmail.cf

As mentioned before this is not the recommended way. Change the m4 files instead and use **m4** to generate `sendmail.cf`.

Exim

Exim is a (MTA) developed at the University of Cambridge for use on Unix systems connected to the Internet. Exim can be installed instead of Sendmail, although the configuration of Exim is quite different. See exim.org for more detailed information.

The *exim4-config_files* man page contains descriptions for each of them. Exim comes with an `exim.conf` template. Just edit this config file for your environment. The *Exim Wiki on GitHub* provides additional configuration help, FAQ, etc.

Postfix

Postfix is another, widely adopted, MTA. It focusses on speed, ease of administration and security. The author of the software, Wietse Venema, was dependant on long computer calculations during his study. It was during that time that he developed a habit of creating software that would be as fault-free and fault-resistant as possible. Only if he could trust the software to run for two weeks straight without errors, there would be no necessity to sleep on the floor for those two weeks while babysitting it. This coding habit stuck with the author for the years to come. When the Postfix program was first written, it was done with the same habit as those earlier study assignments. This habit then became a coding philosophy. By writing the code in a structured manner, only adding neccessary bits and making sure execution flows as efficient as possible, the resulting software became fast, easy to administer and secure. Today, Postfix is still under active development. Many people have contributed to it, and many still do. Despite this common effort, the original author still assures that only necessary code gets added. And that it is added in a way that does not interfere with the philosophy that has been part of Postfix from the start. This approach, in combination with the way that Postfix consists of various small programs working together instead of one big program, makes Postfix a very attractive alternative to other MTA solutions.

At the time of writing Postfix version 3.1 is the latest stable release. Postfix version 3.2 is (still) considered experimental. Experimental Postfix releases can be recognized by their names. These will carry a date in them. An example of an experimental release is the following tarball: `postfix-3.2-20161204.tar.gz`. Stable Postfix versions only show (major and minor) version numbers without the date at the end. Due to its wide adoption, most Linux distributions offer Postfix. However there is a wide variety in versions being deployed. According to its website http://www.postfix.org, versions prior to Postfix 2.10 are End-Of-Support (EOS). CentOS 6.8 still ships with Postfix 2.6 though. Debian 7 did not get beyond Postfix 2.9. More recent Linux distribution versions like CentOS 7 and Debian 8 ship with Postfix >2.10, while some bleeding edge Linux distribution versions offer Postfix >3.1. There is absolutely no certainty that more recent versions of Postfix will have fewer bugs than older versions of Postfix. It is regarded as a 'best practice' to be aware of which version of Postfix you are exposing to the Internet, while at the same time keeping up to date about known Postfix vulnerabilities. There is no reason to restrict this best practice to Postfix alone though.

To address the Postfix objectives that are defined for the LPIC-2 exam, the following will provide some examples. These asume you are using Postfix installed from packages on either a Debian-based or a Red Hat based Linux distribution. There may still be circumstances that require Postfix to be build from source. That subject is beyond the scope of this chapter but the topic in general is covered by LPIC objective 206.1.

Postfix main.cf file format

The default location for the postfix configuration files is `/etc/postfix`. Amongst others there are two main ones that determine Postfix behavior: `main.cf` and . By default, the main.cf configuration file specifies a subset of all the parameters that control the operation of the Postfix mail system. Parameters not explicitly specified are left at their default values. Refer to the `postconf(5)` manpage for a full listing of all parameters. The `main.cf` file contains useful comments as well as references to other documentation sources at the top. On Debian-based systems, the `/etc/postfix/main.cf` file you encounter may be a stripped-down version. The `/usr/share/postfix` directory should then contain a `main.cf.dist` file that is more elaborate. The following line count operation gives an estimate about the differences between the different `main.cf` files on a Debian-based system:

```
user@debian:/usr/share/postfix$ wc -l /etc/postfix/main.cf
43  /etc/postfix/main.cf
user@debian:/usr/share/postfix$ wc -l /usr/share/postfix/main.cf*
18  /usr/share/postfix/main.cf.debian
665 /usr/share/postfix/main.cf.dist
 11 /usr/share/postfix/main.cf.tls
```

The example above demonstrates that the `/usr/share/postfix/main.cf.dist` file holds more than 600 lines, whereas the default `/etc/postfix/main.cf` file only holds 43. Keeping the main.cf file clean and tidy is a good habit. It is also a good habit to use a pager to read through the more elaborate `main.cf.dist` file on Debian-based systems. Red Hat based systems keep well documented configuration files within the `/etc/postfix` directory.

By default, postfix will only read configuration files from the `/etc/postfix` directory. Other configuration directories may be specified by using the `alternate_config_directories` directive in `/etc/postfix/main.cf`. Red Hat based distributions may have `/etc/postfix/dynamicmaps.cf.d` and `/etc/postfix/postfix-files.d` directories. These directories exist to ease the life of the Postfix package creators and should be left alone unless you are really sure about what you are doing.

Note
The postfix configuration directory `/etc/postfix` is "hard-coded" in compilation. Chapter 4.6 of the following document explains how to change this value when building Postfix from source: *Postfix Installation From Source Code*

The general format of the main.cf file is as follows:

- Each logical line is in the form **parameter = value**. Whitespace around the "=" is ignored, as is whitespace at the end of a logical line.
- Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a "#".
- A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.
- A parameter value may refer to other parameters.
- When the same parameter is defined multiple times, only the last instance is remembered.
- Otherwise, the order of `main.cf` parameter definitions does not matter.

Also see *Postfix Configuration Parameters* as an alternative to the **postconf** man page. The remainder of that document is a description of all Postfix configuration parameters. Default values are shown after the parameter name in parentheses. These default values can also be looked up with the **sudo /usr/sbin/postconf -d** command. Beware though, since this command will produce 877 lines of output on a Postfix 3.1 instance. Because it is an administrative command and usually located within `/usr/sbin`, **sudo** or a shell with root privileges should be used to invoke this command.

One of the "best practices" to use with Postfix is to disable the **SMTP VRFY** command on a publicly accessible mail server. This VeRiFY command can be abused by an attacker to enumerate valid user accounts or email adddresses as follows:

```
user@mailserver:~$ telnet mailserver 25
Trying ::1...
Connected to mailserver.
Escape character is '^]'.
220 localhost ESMTP Postfix (Linux/GNU)
EHLO localhost
250-localhost
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
VRFY root
252 2.0.0 root
VRFY wodan
550 5.1.1 <wodan>: Recipient address rejected: User unknown in local recipient table
VRFY postfix
252 2.0.0 postfix
```

The 200-category response codes expose valid recipients. Just like HTTP servers do, a SMTP server can respond with status codes. It is recommended to be familiar with the most common response codes. This will aid when debugging system performance or issues. It is usually not necessary to know the difference between 220 and 250 at first sight. But the difference between 2xx and 5xx status codes should be noticable and familiar. The VRFY command is enabled on Postfix by default. By enabling or adding the following line to `main.cf` the VRFY feature is disabled:

```
disable_vrfy_command = yes
```

Let's try the **VRFY root** command from above again after both having disabled the VRFY command and having reloaded Postfix:

```
user@mailserver:~$ telnet mailserver 25
Trying ::1...
Connected to localhost.
Escape character is '^]'.
220 localhost ESMTP Postfix (Linux/GNU)
EHLO localhost
250-localhost
250-PIPELINING
250-SIZE 10240000
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
VRFY root
502 5.5.1 VRFY command is disabled
VRFY wodan
502 5.5.1 VRFY command is disabled
VRFY postfix
502 5.5.1 VRFY command is disabled
```

By disabling the possibility to distinguish between valid (2xx) and invalid (5xx) email recipients, the Internet has become a safer place.

The Postfix developers recommend to change no more than 2-3 parameters at a time, and test if Postfix still works after every change. After making changes to `main.cf` you need to reload postfix. Depending on the Linux distribution used, this can be accomplished using one of the following commands using **sudo** or a shell with root privileges: **postfix reload** or **systemctl restart postfix** or even **/etc/init.d/postfix reload**.

Postfix master.cf file format

The Postfix mail system follows a modular approach by implementing a small number of (mostly) client commands that are invoked by users, and by a larger number of services that run in the background. Postfix services are implemented by daemon processes. These run in the background and are controlled by the master process. The `master.cf` configuration file defines how a client program connects to a service, and what daemon program runs when a service is requested.

The general format of the `master.cf` file is as follows:

- Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a "#".
- A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.
- Each logical line defines a single Postfix service. Each service is identified by its name and type as described below. When multiple lines specify the same service name and type, only the last one is remembered. Otherwise, the order of master.cf service definitions does not matter.

Each logical line consists of eight fields separated by whitespace. These are described below in the order as they appear in the master.cf file. Where applicable a field of "-" requests that the built-in default value be used. For boolean fields specify "y" or "n" to override the default value.

Chroot option (default: Postfix >= 3.0: n, Postfix <3.0: y) - Whether or not the service runs chrooted to the mail queue directory (pathname is controlled by the *queue_directory* configuration variable in de `main.cf` file).

Postfix preparations

Before postfix can be used, it needs to know:

- what domains to receive mail for (Section 11.1.11.3.1)
- which domain name to use for outbound mail (Section 11.1.11.3.2)
- which domain(s) postfix is allowed to relay mail for (Section 11.1.11.3.3)
- what delivery method to use (Section 11.1.11.3.4)

myorigin

The `myorigin` parameter specifies the domain that appears as the "From:" domain in outgoing email. The `myorigin` option is subject to other options, which define whether the "From:" domain gets appended or replaced by the `myorigin` value. Refer to the `main.cf` file or **postconf** man page for details. By default, the `myorigin` value is defined as `$myhostname`. Changing the `myorigin` value to the configured domain name can be done as follows:

```
myorigin = $mydomain
```

The `$myhostname` or `$mydomain` are replaced by postfix with the hostname or domain of the server it is running on.

mydestination

Postfix needs to know for which domain(s) it will receive mail. The parameter `mydestination` is used for this. More than one domain may be specified. The domain names can be separated using whitespace or a comma. Also, a pattern can be used to point to a lookup table (hash, btree, nis, ldap or mysql).

```
mydestination = $mydomain, localhost.$mydomain, hash:/etc/postfix/moredomains
```

Note
You have to include `$mydomain` when the server is used as a mail server for the entire domain.

relay_domains

The default configuration of postfix will try to deliver incoming mail to *authorized* destinations only. The `relay_domains` parameter controls for which domains postfix will accept and *forward* emails.

```
relay_domains =   (safe: never forward mail from strangers)
```

```
relay_domains = $mydomain (forward mail to my domain and subdomains)
```

relayhost

By default postfix tries to deliver directly to the internet depending on the domain name of the destination address in the mail message. Using the `relayhost` parameter we can specify to use another SMTP server as relay:

```
relayhost =
```

This is the default, direct delivery to the internet, or using another ISP SMTP server:

```
relayhost = mail.example.com
```

Logging

Postfix uses the syslog daemon for its logging. When `/etc/syslog.conf` is configured like in the example below, Postfix log events are written to `/var/log/maillog`. Error messages are, in the example below, redirected to the console.

```
mail.err    /dev/console
mail.debug  /var/log/maillog
```

Tip
Using **egrep '<reject|warning|error|fatal|panic>' /var/log/maillog** will help you to find any problems postfix encountered. There are also third party utilities like **pflogsumm** that can generate statistics out of Postfix logging.

---

virtual domains

Generally a postfix server is the final destination for a limited number of domains. But postfix can also be configured to handle mail for additional domains which are different from, for example, the domain in which the postfix server is located. These destinations are called virtual hosts. Using the `virtual_alias_domains` parameter we can specify for which virtual hosts we wish to receive mail. The format of the parameters is the same as in the samples above. Separate multiple virtual hosts using a space or a comma. Also a link to a (hashed) file on disk is possible:

```
virtual_alias_domains = example.com, snow.nl, unix.nl
```

or when using a hashed file (using the **postmap** utility):

```
virtual_alias_domains = hash:/etc/postfix/virtual
```

The content of `/etc/postfix/virtual` can be:

```
postmaster@example.com  peter
info@snow.nl    gerda
sales@example.com petra
@example.com    jim
```

In the above example peter receives the postmaster@example.com email. Gerda receives the info@snow.nl email and the sales@example.com goes to petra. The last line is a "catch all" rule, all email for example.com without a valid destination goes to jim.

Note
Use **postmap /etc/postfix/virtual** to create the hashed file and issue a **postfix reload** afterwards.

Sendmail emulation layer commands

Since Sendmail has been the de facto mail delivery standard on Unix-like systems for years, replacements like Postfix have felt compelled to implement sendmail emulation layer commands in order to maintain compatibility with outside programs. That is to say that certain commands that were originally included with the sendmail package are available with Postfix as well.

Mailq **mailq** is available on most systems to check the mail queue. It is equivalent to **sendmail -bp**, which works with Postfix too. Its native alternative is **postqueue -p**.

Newaliases **newaliases** is required to generate a binary aliases file with both sendmail and postfix. Both employ `/etc/aliases` as the default input file.

On Postfix systems **man sendmail** will provide more details on the "Postfix to Sendmail compatibility interface".

/var/spool/mail

Specifies the default mail drop directory. By default all mail is delivered to the `/var/spool/mail/<username>` file.

LPIC objective 208.2 covers HTTP over SSL/TLS. The SMTP protocol is similar to the HTTP protocol in that both protocols use plain text network commands. This comes in handy when debugging, because it allows for command execution through telnet- or netcat-like software. But it does not come in handy when transferring sensitive information across a range of systems. Due to the nature of the SMTP protocol, multiple machines may be involved to transfer an e-mail message from the source to the final destination. And there is no way of excluding the possibility that a message has been altered during transfer. This is where encryption comes in. By using encrypted communication, at least some form of confidentiality and authenticity can be added to SMTP transactions. Just like Apache needs to be set up properly for HTTPS transactions, Postfix needs to be configured for use with TLS as well. Unlike Apache, we leave out the SSL acronym and only refer to TLS for Postfix. The following paragraphs will explain more about TLS-specific Postfix directives and about configuring Postfix for use with TLS.

At the time of writing, TLSv1.2 is the latest TLS version. TLSv1.3 has reached DRAFT status, and it will be a matter of time before it hits the streets. There will be some noteworthy changes between TLSv1.2 and TLSv1.3. Up to TLSv1.2, a ciphersuite consisted of an authentication algorithm, an encryption algorithm, a message digest algorithm and a key exchange algorithm. These four algorithms work together and are defined by their acronyms. From TLSv1.3 onward, the ciphersuite will consist of only the encryption and message authentication algorithm.

The following paragraphs are ment to assist you in getting Postfix with TLS up and running. Encryption is an ever expanding subject, and it is recommended to take the time to read and understand the available documentation with regard to Postfix and the (proper) use of TLS. Postfix comes with extensive documentation that may be found within the `/usr/share/doc/postfix` directory (on Debian-based systems, you might have to install the `postfix-doc` package for completeness as is explained on the bottom of this page). The `TLS_README` document is definitely worth reading. It can be viewed using a pager like **less** or first expanded using a utility like **zcat** if the file is compressed. The `TLS_README.txt` document is also available on the Postix website http://www.postfix.org.

When creating a keypair for use with Apache, it may be preferred to use an *encrypted* private key. An encrypted key needs to be decrypted using a password before it can be used. OpenSSL will encrypt the private key during creation by default, unless specified otherwise. Postfix requires the private key used for TLS encrypted communication to be *unencrypted*, in other words without a password. A private key may be *stripped* from its password by reading, importing and exporting the key with OpenSSL. In the following example though, the `-nodes` option is used with OpenSSL during creation time of the key. This will prevent OpenSSL from encrypting the private key in the first place.

```
sudo openssl req -nodes -x509 -newkey rsa:2048 \
-keyout postfixkey.pem -out postfixcert.pem \
-days 356
```

The *self-signed* certificate and private key created with the one-liner above can be used for postfix. Postfix demands the private key file is owned by, and readable only by root. The certificate file may be world-readable. Because the example above uses the RSA algorithm, the following directives are used to refer to these files:

```
smtpd_tls_cert_file=postfixcert.pem
smtpd_tls_key_file=postfixkey.pem
```

Note
When declaring these directives, take special care to differentiate between `smtpd_tls_` directives and `smtp_tls` directives. The latter are used for client authentication.

The certificate file must be in the PEM-format. This is the OpenSSL standard export format, so no additional flags should be required. In the example above, the RSA algorithm is used. Instead of RSA, DSA could also be used as the encryption algorithm. When using the DSA algorithm, the key needs to be generated using the `-t dsa` option and the directives for the certificate file and key slightly differ from their RSA counterparts:

```
smtpd_tls_dcert_file=postfixcert.pem
smtpd_tls_dkey_file=postfixkey.pem
```

ECDSA is another valid encryption algorithm for use with Postfix. But because many mailservers still use OpenSSL 0.9.8 versions that lack the proper ECDSA implementation, the use of ECDSA for Postfix with TLS is currently not recommended. Currently RSA is the recommended encryption algorithm to use with TLS for Postfix. However if you do want to use ECDSA keys, the key must be generated using the `-t ecdsa` option and the certificate and key files have to be specified using the following directives:

```
smtpd_tls_eccert_file=postfixcert.pem
smtpd_tls_eckey_file=postfixkey.pem
```

The OpenSSL example above creates a *self-signed* certificate. This is a way of getting the Postfix STARTTLS functionality working "quick and dirty". But in terms of authenticity, self-signed certificates may not be validated by many public SMTP servers. An alternative is to create a key and CSR, and sign the CSR using your own CA. This is demonstrated in the Apache chapter. By using certificates issued by your own CA, you could let your mailservers validate each other. Yet another option is to create a key and CSR, and have the CSR signed by a public CA. That way, public SMTP servers should be able to validate the certificate offered by your Postfix server. When going this route, it may be necessary to add additional root CA references to complete the certificate chain. Additional root CA files may be configured by using either the `smtpd_tls_CAfile` or `smtpd_tls_CApath` directives. The file directive should point to a (you guessed it) file holding the necessary root CA information. When concatenating several certificates in to one file, be aware that the files are read "bottom to top". The `smtpd_tls_CApath` directive should point to a directory holding the necessary file(s).

Postfix uses *opportunistic encryption* by default. This means that a connecting system will try to create an encrypted connection at first. However, if this fails for any reason (say, due to an invalid certificate) then the system will continue to perform all SMTP commands without encryption. This behavior is determined by the value of the `smtpd_tls_security_level` directive. By changing the value of this directive from `may` to `encrypt`, unencrypted sessions are prevented. Be aware though! This directive may prevent the Postfix server from performing *any* SMTP transactions anymore if TLS sessions cannot be initiated.

Note
Despite all the good intentions from various public Certificate Authorities, the Certificate Authority system is kind of flawed by design. After all, ANY Certificate Authority may issue certificates for ANY service on ANY host. The chapter on DNS mentions DANE TLSA records to combat this issue. When DNSSEC has been properly configured and TLSA records are supported, it is recommended to create 301 or 311 TLSA records for a given TLS-capable Postfix server. By shifting the trust from the CA trust store with over 1300 CA root certificates to the authenticity and integrity of DNSSEC, it becomes less of a problem to depend on self-signed certificates. If multiple domains are served on the same Postfix server, it is recommended to publish a digest for every seperate domain (and therefore certificate) in DNS. The `smtpd_tls_security_level` value can be set to either `dane` or `dane-only` to handle TLSA records. Refer to the `TLS_README` file for details.

Directives and options Postfix has been written with many "what if..." fall-back scenarios in mind. The result is a very resilient piece of software. Being a result of good coding practices, Postfix uses a number of mechanisms to determine if circumstances are favourable or not. If system resources become scarse, the Postfix system will adapt as is deemed suitable regarding the circumstances. When configuring Postfix to use TLS encryption, there are some directives that deserve additional attention. When configured correctly, directives should complement each others functionality. When configured inconsistently or incorrectly, directives may conflict with each others functionality. It is advised to pay additional attention to directives that are involved when configuring the allowed protocols and cipher suites. Modern Postfix versions exclude the use of SSLv2. Because version upgrades may change the default protocols and allowed cipher suites, there is no harm in configuring these yourself. By declaring the following directives in `main.cf`, these options will stay constant despite Postfix version upgrades over time.

```
smtpd_tls_protocols
smtpd_tls_mandatory_protocols
smtpd_tls_ciphers
smtpd_tls_mandatory_ciphers
smtpd_tls_exclude_ciphers
```

By disabling the `aNULL` ciphers, use of *anonymous* ciphers is prevented. The use of anonymous ciphers is one of the fall-back scenarios that helps Postfix in providing availability, at the expense of accountability.

As mentioned before, the TLS functionality in Postfix can be configured using the `smtpd_tls_security_level` directive. According to RFC 2487, this directive *should* be set to the value `may`. This will result in Postfix announcing the availability of `STARTTLS` to connecting clients, without demanding its use. If you want to enforce the use of `STARTTLS`, the value of `smtpd_tls_security_level` should be set to `encrypt` instead. Again, be aware that this may result in degraded server compatibility towards other systems. By setting the directive value to `none`, the Postfix server will refrain from announcing the `STARTTLS` method at all. When a connecting system tries to initiate `STARTTLS`, the Postfix server will reply with a `502 5.5.1 Error:command not implemented` message.

When enabling TLS for Postfix, various additional configuration directives become available. Most directives starting with `smtpd_tls_` or `smtp_tls_` are not significant for the working of Postfix unless TLS is enabled. One of these directives is the `smtpd_tls_loglevel` directive. This directive may be configured by setting a value from `0-4`. It is not recommended to use a setting above `2`, except for debugging purposes. A value of `0` will disable logging of TLS events. A value of `1` will log a summary message on TLS handshake completion. A value of `2` will also log levels during TLS negotiation. A value of `3` will log all of the above plus hex and text dumps of the TLS transaction. A value of `4` eventually will log all of the above but will not stop dumping after the TLS transaction has been processed. It will continue to dump the entire SMTP transmission after the STARTTLS command. Use with caution!

On most Linux distributions, additional documentation regarding Postfix can be found in the `/usr/share/doc/postfix` directory. On Debian-based systems, this requires installing the `postfix-doc` package. Without the addition of this package, the contents of that directory will be very shallow.

Note
Postfix 3.x makes TLS easier by incorperating the **postfix-tls**. While reading up, you might also want to check out what **tlsmgr** and **tlsproxy** do.

Managing E-Mail Delivery (211.2)

Candidates should be able to implement client e-mail management software to filter, sort and monitor incoming user email.

Key Knowledge Areas

Understanding of Sieve functionality, syntax and operators

Use Sieve to filter and sort mail with respect to sender, recipient(s), headers and size

Awareness of procmail

Terms and Utilities

- Conditions and comparison operators
- keep, fileinto, redirect, reject, discard, stop
- Dovecot vacation extension

Procmail

Procmail is a email filtering utility that may be used for preprocessing and sorting of incoming mail. It can also be used to sort out email from mailinglists, to filter spam and send auto-replies. Procmail configuration is based on a file placed in the user's homedirectory. It is rarely run from the command line (except for testing purposes) but it's an autonomous program which is normally invoked by MTA's (Mail Transport Agent) like Sendmail or Postfix.

Procmail follows the following scheme for reading its configuration (it reads both):

```
/etc/procmailrc
~/.procmailrc
```

Be careful using the system-wide `/etc/procmailrc`. It is usually read and processed as root. This fact means that a poorly designed recipe in that file could do serious damage. For instance, a typo could cause Procmail to overwrite an important system binary rather than use that binary to process a message. For this reason, you should keep system-wide Procmail processing to a minimum and instead focus on using `~/.procmailrc` to process email using individual accounts.

Sieve

Sieve is a scripting language that may be used to preproces and sort incoming email. It can also be used to sort out email from mailinglists, to filter spam and send auto-replies. To use sieve it should first be configured on the email servers. In this setup postfix is used to deliver email to the Dovecot local delivery agent. In the file `main.cf` the *mailbox_command* option should be configured to use the local delivery agent.

```
mailbox_command = /usr/lib/dovecot/dovecot-lda -a "$RECIPIENT"
```

The next step is enabling sieve support in dovecot. In the configuration file `15-lda.conf` the following options should be configured:

```
lda_mailbox_autocreate = yes

lda_mailbox_autosubscribe = yes

protocol lda {
    mail_plugins = $mail_plugins sieve
}
```

The configuration option: *lda_mailbox_autocreate* enables dovecot to create a mailbox if this action is initiated by a rule in sieve. The option: *lda_mailbox_autosubscribe* subscribes a user to a specific mailbox if this is initiated by an auto created action for a specific mailbox. The option: *mail_plugins* enables the sieve scripting module in dovecot. The configuration file `90-sieve.conf` needs the following adjustments:

```
plugin {
    sieve = ~/.dovecot.sieve
    sieve_dir = ~/sieve
  sieve_default = /var/lib/dovecot/sieve/default.sieve
  sieve_global_dir = /var/lib/dovecot/sieve
}
```

The *sieve* configuration option is the location where users can save their sieve rules. The *sieve_dir* configuration option is the directory of a user which can hold multiple sieve scripts. The configuration option *sieve_default* is used to execute a default sieve script. If a user has a personal sieve configuration file in its home directory then the global configuration file is overruled. The option *sieve_global_dir* identifies a global directory to contain multiple global sieve scripts. After configuration the services should be restarted with **service postfix restart** and **service dovecot restart**.

Sieve syntax

The *sieve* syntax is easy to understand. It consists of three basic parts: Control, Test and Action commands. The control command controls the flow of the code. They affect how the commands will be carried out. The test command will be used in conjunction with control commands to specify a condition that could lead to an action. The action command will be executed after a condition is evaluated to true. You can also use single-line comments starting with a "#" and multi-line comments starting with "/*" and ending with "*/" to clarify sieve rules. Example:

```
# comment
require ["extension"];

/* This is multi
   line comment */

# if -> control command
if <condition> {
   action1;
   action2;
   ...
   stop; #end processing
```

Sieve also offers an auto-responder functionality by using the `vacation>` extension. Below is an example shown that uses the vacation extension.

```
require ["fileinto", "vacation"];

vacation
  # Reply at most once a day to a same sender
  :days 1
      :subject "Out of office reply"
  # List of additional recipient addresses which are included in the auto replying.
  # If a mail's recipient is not the envelope recipient and it's not on this list,
  # no vacation reply is sent for it.
  :addresses ["j.doe@company.dom", "john.doe@company.dom"]
  "I'm out of office, please contact Joan Doe instead.
Best regards
John Doe";
```

The "vacation" extension provides several options, namely:

- `:days number` - Is used to specify the period where addresses are kept and not responded to (in days).

- `:subject string` - Specifies the subject line attached to any vacation response.
- `:from string` - Specifies an alternate to use in the From field of vacation messages.
- `:addresses string-list` - Specifies additional email addresses to the recipient.
- `:mime` - Specifies arbitrary mime content. For example to specify multiple vacation messages in different languages.
- `:handle string` - Tells sieve to treat two vacation actions with different arguments as the same command for response tracking
- *reason: string* - The actual message

Example:

```
require "vacation";
if header :contains "subject" "lunch" {
         vacation :handle "ran-away" "I'm out and can't meet for lunch";
  } else {
         vacation :handle "ran-away" "I'm out";
  }
```

Control commands

The control commands in sieve are the basic **if**, **else** and **elsif** control statements. If the test condition is evaluated to true then the associated action will be executed. In the control statements the following tagged arguments can be used: **:contains**, **:is**, **:matches**, **:over**, and **:under** as shown in the upcoming sieve examples. The **require** control command is used to declare optional extensions at the beginning of a script (e.g. fileinto) and the **stop** command ends all processing of the script. Example:

```
require "fileinto";
  if header :contains "from" "lottery" {
    discard;
} elsif header :contains ["subject"] ["$$$"] {
          discard;
      } else {
          fileinto "INBOX";
      }
```

Example:

```
if header :contains "subject" "money" {
   discard;
   stop;
}
```

Test commands

The control commands as stated in the previous section can support different test commands, namely: **address**, **allof**, **anyof**, **exists**, **false**, **header**, **not**, **size** and **true**.

With the address command you can only test if an email adres is in the header. If the **to** header contains "John Doe <john@doe.com>", then the test would evaluate to false. If the **to** address contains "john@doe.com" then the test would be true because only the address is evaluated. Example:

```
require "fileinto";

if address :is "to" "john@doe.com" {
   fileinto "john";
}
```

The **allof** command is a logical “AND” meaning all conditions should be evaluated to true for further action. Example:

```
if allof (header :contains "from" "Bofh", header :contains "to" "abuse")
{
  fileinto "spam";
}
```

The **anyof** command is a logical “OR” meaning ANY condition should be evaluated to true for further action. Example:

```
if anyof (size :over 1M, header :contains "subject" "big file attached")
{
    reject "I don’t want messages that claim to have big files.";
}
```

The **exists** command tests if a header exits with the message. All headers must return true for any action being executed. Example:

```
if exists "x-custom-header"
{
  redirect "admin@example.com";
}
```

The **false** command simply returns false.

The **header** command tests if a header matches the condition set by the argument and evaluates to true. Example:

```
if header :is ["subject"] "make money fast" {
          discard;
    stop;
}
```

The **not** command should be used with another test. This command negates the other test for the action to be taken. The example below means that if the message does NOT contain “from” and “date” then the discard action will be taken. Example:

```
if not exists ["from", "date"]
{
   discard;
}
```

The **size** command is used to specify the message size to be higher or lower than a specified value in order to evaluate the condition to true. The command accepts tagged arguments `:over` and `:under` and you can use M after the specified value for megabytes, K for kilobytes and no letter for bytes. Example:

```
if size :over 500K {
   discard;
}
```

The **true** command simply returns true.

Action commands

The action commands are being executed after a test command is evaluated to true or operate on their own. The action commands are: **keep**, **fileinto**, **redirect** and **discard**. The **keep** action commands causes the message to be saved in the default location. The **fileinto** action command is an optional command and can be used by using **require "fileinto"** control command in the beginning of the script. If the test command is evaluated to true then the message is moved into the defined mailbox. Example:

```
if attachment :matches ["*.vbs", "*.exe"] {
    fileinto "INBOX.suspicious";
}
```

The **redirect** command redirects the message to the address that is specified in the argument without tampering the message. Example:

```
if exists "x-virus-found" {
   redirect "admin@example.com";
}
```

The **discard** command causes the message silently deleted without sending any notification or any other message. Example:

```
if size :over 2M {
  discard;
}
```

Mbox and maildir storage formats

Mbox and maildir are email storage formats. Postfix and Dovecot support the two email storage formats where maildir is the recommended format.

Mbox format

Mbox is the traditional email storage format. In this format there is only one regular text file which serves as the user's mailbox. Typically, the name of this file is `/var/spool/mail/<user name>`. Mbox locks the mailbox when an operation is performed on the mailbox. After the operation the mailbox is unlocked.

Advantages:

- Mbox format is universally supported
- Appending new email is fast
- Searching inside single mailbox is fast

Disadvantages:

- Mbox is known for locking problems
- The mbox format is prone to corruption

Maildir format

Maildir is the newer email storage format. A directory maildir is created for each email user, typically in the users' home directories. Under this maildir directory by default three more directories exist: new, cur and tmp.

Advantages:

- Locating, retrieving and deleting a specific email is fast, particularly when a email folder contains hundreds of messages
- Minimal to no file locking needed
- Can be used on a network file system
- Immune to mailbox corruption (assuming the hardware will not fail)

Disadvantages:

- Some filesystems may not efficiently handle a large number of small files
- Searching text, which requires all email files to be opened is slow

Recipe differences between mbox and maildir for procmailrc

Before copying the recipes from this page into your procmailrc file, remember to adapt them to your particular maildir/mbox format, taking into consideration that the name of maildir folders end in "/". You do not need to lock the file when using maildir format (:0 instead of :0:).

In mbox the format is:

```
:0:
recipe
directory_name
```

While in maildir it would be:

```
:0
recipe
directory_name/
```

Managing Mailbox Access (211.3)

Candidates should be aware of Courier email server and be able to install and configure POP and IMAP daemon on a Dovecot server.

Courier

The Courier mail transfer agent (MTA) is an integrated mail/groupware server based on open commodity protocols, such as ESMTP, IMAP, POP3, LDAP, SSL, and HTTP. Courier provides ESMTP, IMAP, POP3, webmail, and mailing list services within a single, consistent framework. Individual components can be enabled or disabled at will. The Courier mail server now implements basic web-based calendaring and scheduling services integrated in the webmail module.

The Courier mail server uses maildirs as its native mail storage format, but it can also deliver mail to legacy mailbox files as well. By default `/etc/courier` is the sysconfdir. All courier configuration files are stored here. The mail queue can be found at `/var/spool/mqueue`.

Information about the configuration for Courier can be found at: *Courier installation*.

Dovecot

Dovecot is an open source IMAP and POP3 email server for Linux/UNIX-like systems, written with security primarily in mind. Dovecot claims that it is an excellent choice for both small and large installations.

The configuration of Dovecot can be found in `/etc/dovecot.conf` and we need to configure several parameters: authentication, mailbox location, SSL settings and the configuration as POP3 server.

Authentication

Dovecot is capable of using several password database backends like: PAM, BDSAuth, LDAP, passwd, and SQL databases like MySQL, PostgreSQL and SQLite. The most common way is PAM authentication. The PAM configuration is usually located in `/etc/pam.d`. By default Dovecot uses `dovecot` as PAM service name.

Here is an example of `/etc/pam.d/dovecot`:

```
auth    required        pam_unix.so nullok
account required        pam_unix.so
```

The method used by clients to send the login credentials to the server, is configured via the *mechanisms* parameter. The simplest authentication mechanism is *PLAIN*. The client simply sends the password unencrypted to Dovecot. All clients support the *PLAIN* mechanism, but obviously there's the problem that anyone listening on the network can steal the password. For that reason (and some others) other mechanisms were implemented.

SSL/TLS encryption can be used to secure the *PLAIN* authentication mechanism, since the password is sent over an encrypted stream. Non-plaintext mechanisms have been designed to be safe to use even without SSL/TLS encryption. Because of how they have been designed, they require access to the plaintext password or their own special hashed version of it. This means that it's impossible to use non-plaintext mechanisms with commonly used DES or MD5 password hashes. With success/failure password databases (e.g. PAM) it's not possible to use non-plaintext mechanisms at all, because they only support verifying a known plaintext password.

Dovecot supports the following non-plaintext mechanisms: *CRAM-MD5*, *DIGEST-MD5*, *APOP*, *NTLM*, *GSS-SPNEGO*, *GSS-API*, *RPA*, *ANONYMOUS*, *OTP* and *SKEY, EXTERNAL*. By default only the *PLAIN* mechanism is enabled. You can change this by modifying `/etc/dovecot.conf`:

```
auth default {
  mechanisms = plain login cram-md5
  # ..
}
```

Mailbox location

Using the *mail_location* parameter in `/etc/dovecot.conf` we can configure which mailbox location we want to use:

```
mail_location = maildir:~/Maildir
```

or

```
mail_location = mbox:~/mail:INBOX=/var/mail/%u
```

In this case email is stored in `/var/mail/%u` where "%u" is converted into the username.

SSL

Before Dovecot can use SSL, the SSL certificates need to be created and Dovecot must be configured to use them.

Dovecot includes a script **mkcert.sh** to create self-signed SSL certificates:

```
#!/bin/sh

# Generates a self-signed certificate.
# Edit dovecot-openssl.cnf before running this.

OPENSSL=${OPENSSL-openssl}
SSLDIR=${SSLDIR-/etc/ssl}
OPENSSLCONFIG=${OPENSSLCONFIG-dovecot-openssl.cnf}

CERTDIR=$SSLDIR/certs
KEYDIR=$SSLDIR/private

CERTFILE=$CERTDIR/dovecot.pem
KEYFILE=$KEYDIR/dovecot.pem

if [ ! -d $CERTDIR ]; then
  echo "$SSLDIR/certs directory doesn't exist"
  exit 1
fi

if [ ! -d $KEYDIR ]; then
```

```
  echo "$SSLDIR/private directory doesn't exist"
  exit 1
fi

if [ -f $CERTFILE ]; then
  echo "$CERTFILE already exists, won't overwrite"
  exit 1
fi

if [ -f $KEYFILE ]; then
  echo "$KEYFILE already exists, won't overwrite"
  exit 1
fi

$OPENSSL req -new -x509 -nodes -config $OPENSSLCONFIG -out $CERTFILE -keyout $KEYFILE - ↩
    days 365 || exit 2
chmod 0600 $KEYFILE
echo
$OPENSSL x509 -subject -fingerprint -noout -in $CERTFILE || exit 2
```

The important SSL configuration options can be found in the file: `conf.d/10-ssl.conf`. To enable encryption of the data in transit between a client and a Dovecot server the following changes should be made.

```
ssl = required
```

This configuration option requires that the client is using SSL/TLS as transport layer mechanism. Authentication attempts without SSL/TLS will cause authentication failures. Another important configuration option to enable SSL/TLS is the configuration of the SSL/TLS key and the SSL/TLS certificate. The certificates in this example are auto generated by the installation of Dovecot.

```
ssl_cert = </etc/dovecot/dovecot.pem
ssl_key = </etc/dovecot/private/dovecot.pem
```

The preferred permissions of the certificate is 0440 (world readable). The certificate is offered to clients. The permissions of the key should be 0400 with uid/gid 0. It should only be readable by the root user. If the key file is password protected the password can be configured in the configuration file by changing the `ssl_key_password` option. Since the SSL and TLSv1 protocols are vulnerable to multiple attacks like POODLE (Padding Oracle On Downgraded Legacy Encryption) those protocols should be disabled.

```
ssl_protocols = !SSLv2 !SSLv3 !TLSv1
```

Another key feature of configuring encryption is determine the cipher suite that should be used by Dovecot. The cipher suite defines the allowed ciphers offered by the server by initiating a secured connection with the client. You should keep in mind that the mail user agent should support the cipher suite that is configured on the server otherwise it is not possible to establish a secure connection. An example of a cipher suite is displayed below:

```
ssl_cipher_list = AES256+EECDH:AES256+EDH
```

The cipher AES256+EECDH means that the cipher is using authenticated Ephemeral Elliptic Curve Diffie Hellman key agreement protocol. This protocol is used the share a secret over an insecure channel. This key can be used to encrypt and decrypt communications by using a symmetric encryption protocol which is AES256 bits in this configuration. The cipher AES256+EDH is almost the same as AES256+EECDH. This cipher is not using elliptic curves but RSA algorithm. Another option that should be configured is:

```
ssl_prefer_server_ciphers = yes
```

This option prefers the ciphers that are in the configured on the server in favour of the ciphers from the client. This configuration option avoids so called downgrade attacks. This attack is performed by a man in the middle attack and removes the strong crypto suites to initiate only weak ciphers from the client. The attacker can attack the weak ciphers with main purpose to decrypt encrypted traffic. Another important configuration option is:

```
ssl_dh_parameters_length = 2048
```

This option configures Diffie-Hellman key exchange to 2048-bit keys. Recently the Logjam vulnerability was published. This attack is related to cipher suite down grade attacks. An attacker can downgrade a TLS connection to use 512-bit DH cryptography. On a linux client the supported cipher suite by first list the shared library (e.g. openssl or gnutls) and then listing the supported ciphers by using one of the command **openssl ciphers** or with **gnutls-cli -l**. If a cipher is not supported by the mail user agent for example mutt, it will display an error e.g.

```
gnutls_handshake: A TLS fatal alert has been received.(Handshake failed)
```

. After the SSL/TLS configuration the imaps and pop3s listener should be configured. The listeners can be configured in the file: `10-master.conf`.

```
service imap-login {
    inet_listener imap {
        port = 0
   #port = 143
 }
    inet_listener imaps {
        port = 993
 }
}

service pop3-login {
    inet_listener pop3 {
        port = 0
   #port = 110
 }
    inet_listener pop3s {
        port = 995
   }
}
```

If the listener port is set to 0 the pop3 and imap service are not running on the server. Only the secure versions of the protocol are enabled. After configuration of the dovecot the dovecot server should be restarted. This can be initiated by the command: **service dovecot restart**. Verify if pop3s and imaps service is listening on the appropriate port by using the command:

```
# netstat -anp |egrep '993|995'
tcp    0   0 0.0.0.0:993      0.0.0.0:*    LISTEN      3515/dovecot
tcp    0   0 0.0.0.0:995      0.0.0.0:*    LISTEN      3515/dovecot
tcp6   0   0 :::993           :::*         LISTEN      3515/dovecot
tcp6   0   0 :::995           :::*         LISTEN      3515/dovecot
```

As you can see pop3s and imaps are listening their configured ports and ready to use.

POP3 server

Although Dovecot is primarily designed as IMAP server, it works fine as POP3 server but it isn't optimized for being that. The POP3 specification requires that sizes are reported exactly and using *Maildir* the linefeeds are stored as plain LF characters. Simply getting the file size therefore returns a wrong POP3 message size.

When using *mbox* instead of *Maildir*, the index files are updated when a POP3 starts and includes all messages. After the user has deleted all mails, the index files again get updated to contain zero mails. When using Dovecot as a POP3 server, you might want to consider disabling or limiting the use of the index files using the **mbox_min_index_size** setting.

Questions and answers

E-Mail services

1. *How can you check whether it is possible to reach a certain SMTP server?*

   Use **telnet SMTP-server 25** and wait for a connected response. SMTP session using a telnet connection [316]

2. *Postfix uses two configuration files: `postfix.cf` and `master.postfix`. Correct?*

   No, those files are: `main.cf` and `master.cf`. Postfix [320]

3. *Which parameter does Postfix use in order to learn for which domain(s) to receive mail?*

   The parameter to be used is **mydestination**. Multiple domainnames can be seperated using a whitespace, a comma or both. Configuring Postfix [323]

4. *What is the purpose of using virtual domains?*

   In order to configure Postfix to handle mail for additional domains besides its own virtual hosts can be added. Virtual Domains [324]

5. *What is procmail?*

   Procmail is a mail filtering utility and is used for pre-processing and sorting incoming mail. Procmail [328]

6. *How would you get an overview of all flags that **procmail** recognizes?*

   procmail -h.

7. *What does the bang character (!) mean at the beginning of the last line of a recipe?*

   It means that the message that matches the previous condition(s) has to be forwarded to another mailbox.

8. *Why is it necessary with Courier to create system aliases?*

   Courier does not deliver mail to root (for security reasons) and therefore system aliases (at least for root) need to be created.

9. *Which is the most common way to implement authentication for Dovecot?*

   The most common way is using PAM authentication. Dovecot authentication [333]

10. *Why would you configure and limit the use of the index files using **mbox_min_index_size** when using Dovecot as a POP3 server?*

    This is a workaround in order to get updated on deleted mail messages when using *mbox* instead of *maildir*. Dovecot POP3 server [336]

Chapter 12

System Security (212)

This topic has a total weight of 14 points and contains the following objectives:

Objective 212.1; Configuring a router (3 points) Candidates should be able to configure a system to perform network address translation (NAT, IP masquerading) and state its significance in protecting a network. This objective includes configuring port redirection, managing filter rules and averting attacks.

Objective 212.2; Securing FTP servers (2 points) Candidates should be able to configure an FTP server for anonymous downloads and uploads. This objective includes precautions to be taken if anonymous uploads are permitted and configuring user access.

Objective 212.3; Secure shell (SSH) (4 points) Candidates should be able to configure and secure an SSH daemon. This objective includes managing keys and configuring SSH for users. Candidates should also be able to forward an application protocol over SSH and manage the SSH login.

Objective 212.4; Security tasks (3 points) Candidates should be able to receive security alerts from various sources, install, configure and run intrusion detection systems and apply security patches and bugfixes.

Objective 212.5; OpenVPN (2 points) Candidates should be able to configure a VPN (Virtual Private Network) and create secure point-to-point or site-to-site connections.

Sources of information: https://openvpn.net, IPTables, OpenSSH, FTP

Configuring a router (212.1)

Candidates should be able to configure a system to perform network address translation (NAT, IP masquerading) and state its significance in protecting a network. This objective includes configuring port redirection, managing filter rules and averting attacks.

Key Knowledge Areas

Network Address Translation (NAT)

iptables configuration files, tools and utilities

Tools, commands and utilities to manage routing tables

Private address ranges

Port redirection and IP forwarding

List and write filtering and rules that accept or block datagrams based on source or destination protocol, port and address

Save and reload filtering configurations

Awareness of ip6tables and filtering

Terms and Utilities

- `/proc/sys/net/ipv4`
- `/etc/services`
- **iptables**

Private Network Addresses

Why do Private Network Addresses exist? It has been common practice to assign globally-unique addresses to all hosts that use IP addresses. 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)
```

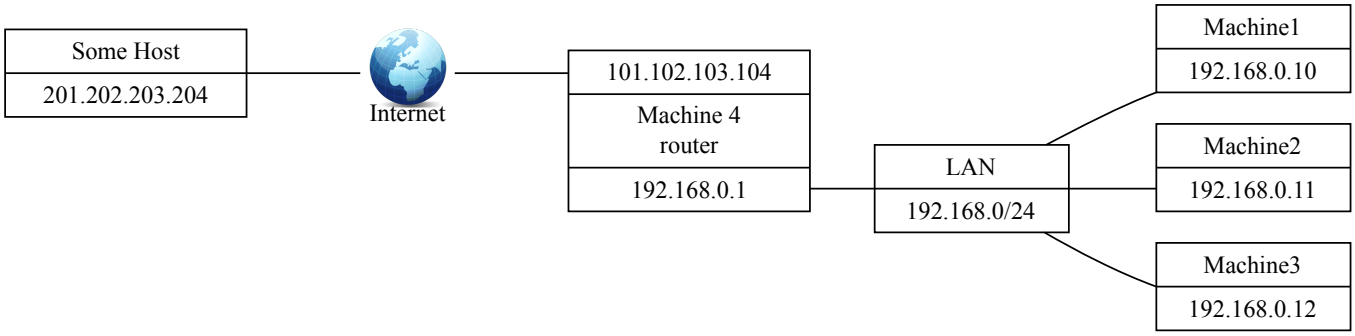
We will refer to the first block as "24-bit block", the second as "20-bit block", and to the third as "16-bit" block. Note that when no subnetting is used (i.e. in pre- Classless Inter-Domain Routing (CIDR) notation) the first block is nothing but a single class A network number, while the second block is a set of 16 contiguous class B network numbers and third block is a set of 256 contiguous class C network numbers.

Even though IPv6 addresses are not likely to run out in the foreseeable future, the need for allocating private addresses has been recognized. RFC4193 describes address block fc00::/7, which is the approximate counterpart of the IPv4 private addresses described above.

In addition to private IP addresses, IPv6 re-introduces the concept of link-local addresses, valid only for communications within the network segment (link) or the broadcast domain that the host is connected to. Routers do not forward packets with link-local addresses, because they are not guaranteed to be unique outside their network segment. In IPv4, the network range 169.254.0.0/16 was reserved for interfaces to allocate an IP address to themselves automatically. In practice, finding an IP address in this range on an interface generally means that DHCP allocation has failed, as link-local addressing is not generally used in IPv4 networks.

In IPv6 networks, interfaces *always* allocate a link-local address in addition to potentially other configured or allocated IPv6 addresses. Therefore, IPv6 interfaces usually have more than one address. Link-local address are an integral part of the IPv6 protocol standard to facilitate neighbour discovery (NDP) and allocating globally unique IP addresses using DHCP6. Interfaces configured for IPv6 use part of their MAC address as a means to create a (hopefully) unique link-local address in the fe80::/64 range.

Network Address Translation (NAT)



The figure above displays a hypothetical situation which will serve as an example in the following explanation.

This section describes Network Address Translation (NAT), which is a technique to rewrite the source or destination address (or sometimes both) of certain IP traffic. It can be used to enable hosts using a private IP address to communicate with hosts using a globally unique IP address. NAT is primarily an IPv4 concept. IPv6 discourages the use of NAT, because its creators believed it causes more problems than it solves. Under certain circumstances beyond the scope of this book, however, a need to rewrite IPv6 addresses may arise.

"The Firm" has four machines, 1 to 4, which are connected via a network switch 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 network switch 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.example) 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 (routed) 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.

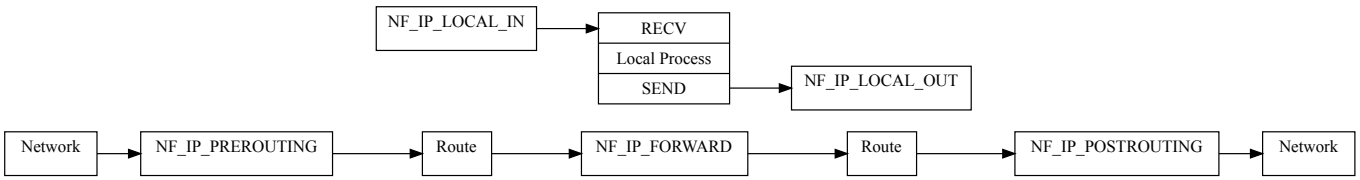
This is in a nutshell how NAT works. For more detailed information consult RFC1631.

The Linux firewall, an overview

Implementation

The Linux firewall is implemented in the kernel (as of version 2.3.15). The *NETFILTER* modules implement the packet filtering rules. The user space application **iptables** is used to configure these rules.

Netfilter "hooks"



As the figure above shows, netfilter supports five different hooks in the protocol stack. These hooks enable us to examine and modify (if necessary) every packet passing through the kernel.

THERE ARE FIVE POSSIBLE ACTIONS:

NF_ACCEPT Continue traversal as normal.

NF_DROP Drop the packet and do not continue traversal.

NF_QUEUE Queue the packet for userspace handling.

NF_REPEAT Call this hook again.

NF_STOLEN Take over (absorb) the packet but do not continue traversal.

Tables and Chains

By default five chains (the netfilter hooks) and three tables are supported. As the figure below shows, certain chains are only valid for certain tables.

		CHAIN				
		PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
TABLE	MANGLE	V			V	
	NAT	V			V	V
	FILTER		V	V	V	

Table 12.1: Valid chains per table

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 intended for the firewall itself. The FORWARD chain is used for all packets that come from outside the firewall and are destined for another machine outside the firewall. These packets must flow through the firewall. 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 incoming packets, before any routing decision has taken place. 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. Note that traffic flowing through the firewall passes the PREROUTING and POSTROUTING chains, whereas traffic originated by the firewall passes the OUTPUT and POSTROUTING chains.

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 to alter incoming packets, before any routing decision has taken place. 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*. These firewalls keep track of established connections by memorizing the source and destination addresses and port numbers (so-called 5-tuples) mostly in order to determine valid return traffic. For protocols that do not use port numbers (e.g. ICMP) other properties are maintained. When using a stateful firewall, firewall rules have to be configured for traffic going one way only, as valid return traffic is passed automatically by a catch-all rule.

The **iptables** option used for connection tracking is the `--state` option.

THE STATE OPTION

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: NEW, ESTABLISHED, RELATED, and INVALID.

MODULES FOR CONNECTION TRACKING

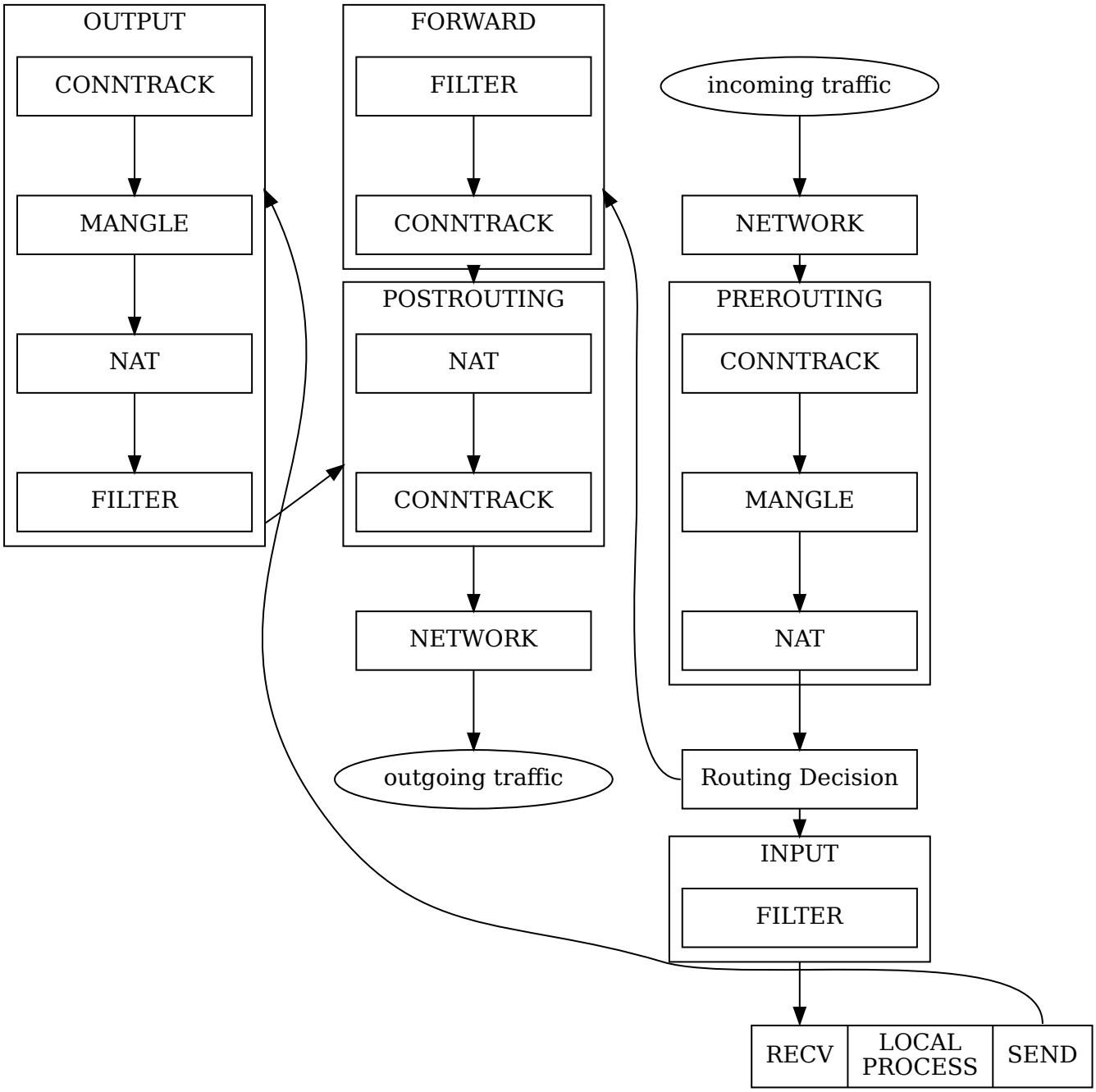
`ip_conntrack` The main connection-tracking code.

`ip_conntrack_ftp` Additional code needed to track ftp connections, both active and passive.

The connection tracking modules have hooks into 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. Packets which do not match any rule in the chain are subject to an implicit target defined as the policy of the chain. Standard targets always present are:

ACCEPT Let the packet through.

DROP Absorb the packet and forget about it.

QUEUE Pass the packet to user space.

RETURN 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.

EXTENDED TARGET MODULES INCLUDED IN MOST LINUX DISTRIBUTIONS

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 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).

FOLLOWING PACKET MATCHING MODULES INCLUDED IN MOST LINUX DISTRIBUTIONS

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.

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 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).

iptables options

-t, --table *table* Table to manipulate (default: "filter"). The tables are as follows:

- **filter** This is the default table (if no -t option is passed). It contains the built-in chains INPUT (for packets destined to local sockets), FORWARD (for packets being routed through the box), and OUTPUT (for locally-generated packets).
- **nat** This table is consulted when a packet that creates a new connection is encountered. It consists of three built-ins: PREROUTING (for altering packets as soon as they come in), OUTPUT (for altering locally-generated packets before routing), and POSTROUTING (for altering packets as they are about to go out).
- **mangle** This table is used for specialized packet alteration. Until kernel 2.4.17 it had two built-in chains: PREROUTING (for altering incoming packets before routing) and OUTPUT (for altering locally-generated packets before routing). Since kernel 2.4.18, three other built-in chains are also supported: INPUT (for packets coming into the box itself), FORWARD (for altering packets being routed through the box), and POSTROUTING (for altering packets as they are about to go out).
- **raw** This table is used mainly for configuring exemptions from connection tracking in combination with the NOTRACK target. It registers at the netfilter hooks with higher priority and is thus called before ip_conntrack, or any other IP tables. It provides the following built-in chains: PREROUTING (for packets arriving via any network interface) OUTPUT (for packets generated by local processes).

-A, --append *chain rule-specification* Append one or more rules to the end of the selected chain.

-D, --delete *chain rule-specification* , -D, --delete *chain rulenum* Delete one or more rules from the selected chain. You can use a rule-specification or a rule number.

-I, --insert *chain* [*rulenum*] *rule-specification* Insert one or more rules in the selected chain as the given rule number.

-R, --replace *chain rulenum rule-specification* Replace a rule in the selected chain.

-L, --list [*chain*] List all rules in the selected chain. This option is often used with the `-n` option for numeric output instead of displaying the output with the host names, network names and service names. This option also shows the default policy of each chain.

-F, --flush [*chain*] Flush the selected chain (all the chains in the table if none is given).

-P, --policy *chain target* Set the policy for packets not matched by any rule in the chain to the given target. This target can be a user-defined chain or one of the special values ACCEPT, DROP or REJECT.

-v, --verbose Verbose output.

--line-numbers When listing rules, add line numbers to the beginning of each rule, corresponding to the rule's position in the chain.

-N, --new-chain *chain* Create a new *user-defined* chain by the given name. There must be no target of that name already.

-X, --delete-chain *chain* Delete the optional *user-defined* chain specified. There must be no references to the chain. If there are, you must delete or replace the referring rules befor the chain can be deleted. If no argument is given, it will attempt to delete every non-builtin chain in the table!

Every chain has a default policy. This can only be changed when the chain is empty. You can see the default policy using the `-L` option. Then flush the chain of all its rules using the `-F table` option. Then set the default policy using the `-P` option.

```
iptables -t filter -L
iptables -t filter -F INPUT
iptables -t filter -P INPUT DROP
```

REJECT is not possible as a default policy. If you still want REJECT as an (implied) last rule then add a REJECT rule yourself as the last rule in the chain.

iptables parameters

The following parameters make up a rule specification (as used in the add, delete, insert, replace and append commands).

[!] -p, --protocol *protocol* The protocol of the rule or of the packet to check. The specified protocol can be one of tcp, udp, udplite, icmp, esp, ah, sctp or the special keyword "all", or it can be a numeric value representing one of these protocols. A protocol name from `/etc/protocols` is also allowed. A "!" argument before the protocol inverts the test. The number zero is equivalent to all.

[!] -s, --source *address* [/*mask*][,...] Source specification. Address can be either a network name, a hostname, a network IP address (with /mask), or a plain IP address. Hostnames will be resolved once only, before the rule is submitted to the kernel. Please note that specifying any name to be resolved with a remote query such as DNS is a really bad idea. The mask can be either a network mask or a plain number, specifying the number of 1's at the left side of the network mask. Thus, a mask of 24 is equivalent to 255.255.255.0. Multiple addresses can be specified, but this will expand to multiple rules (when adding with -A), or will cause multiple rules to be deleted (with -D).

[!] -d, --destination *address* [/*mask*][,...] Destination specification. See also the "source address" parameter above.

-j, --jump *target* This specifies the target of the rule; i.e., what to do if the packet matches it.

[!] -i, --in-interface *name* Name of an interface via which a packet was received.

[!] -o, --out-interface *name* Name of an interface via which a packet is going to be sent.

iptables match extensions

iptables can use extended packet matching modules. These are loaded in two ways: implicitly, when `-p` or `--protocol` is specified, or with the `-m` or `-match` options, followed by the matching module name. Possible are: addrtype, ah, cluster, comment, connbytes, connlimit, connmark, conntrack, dccp, dscp, ecn, esp, hashlimit, helper, icmp, iprange, length, limit, mac, mark, multiport, owner, physdev, pkttype, policy, quota, ratetest, realm, recent, sctp, set, socket, state, statistic, string, tcp, tcpmss, time, tos, ttl, u32, udp and unclean. A few of them are explained her:

-m state This module, when combined with connection tracking, allows access to the connection tracking state of this packet.

[!] --state *state* Here state is a comma separated list of the connection states to match. Possible states are INVALID (meaning that the packet could not be identified for some reason), ESTABLISHED (meaning that the packet is associated with a connection which has seen packets in both directions), NEW (meaning that the packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions), and RELATED (meaning that the packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer, or an ICMP error).

-m time This matches if the packet arrival time/date is within a given range. All options are optional, but are ANDed when specified. It provides the following options:

-m time --datestart *YYYY* [-*MM* [-*DD* [T*hh* [:*mm* [:*ss*]]]]] Only match during the given time, which must be in ISO 8601 "T" notation. The possible time range is 1970-01-01T00:00:00 to 2038-01-19T04:17:07.

-m time --datestop *YYYY* [-*MM* [-*DD* [T*hh* [:*mm* [:*ss*]]]]] Only match during the given time, which must be in ISO 8601 "T" notation. The possible time range is 1970-01-01T00:00:00 to 2038-01-19T04:17:07.

-p udp, --protocol udp These extensions can be used if "--protocol udp" is specified. It provides the following options:

[!] --source-port,--sport *port* [:*port*] Source port or port range specification.

[!] --destination-port,--dport *port* [:*port*] Destination port or port range specification.

-p tcp, --protocol tcp These extensions are loaded if "--protocol tcp" is specified. It provides among other things the following options:

--source-port, --sport [!] port[:port] Source port or port range specification. This can either be a service name or a port number. An inclusive range can also be specified, using the format port:port. If the first port is omitted, "0" is assumed; if the last is omitted, "65535" is assumed. If the second port greater then the first they will be swapped. The flag --sport is a convenient alias for this option.

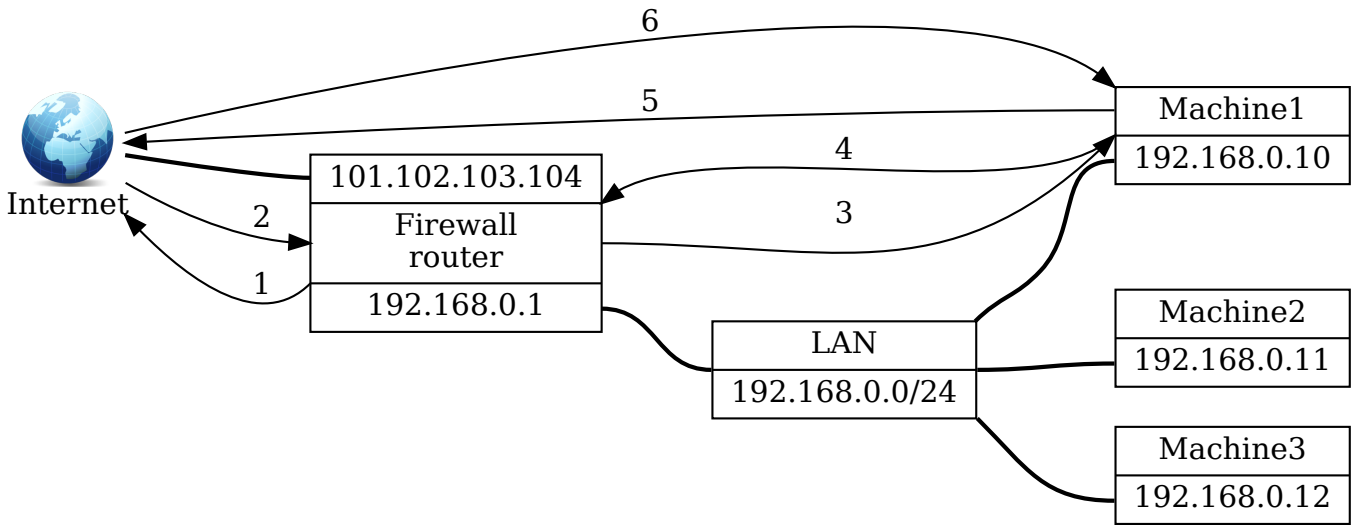
--destination-port, --dport [!] port[:port] Destination port or port range specification. The flag --dport is a convenient alias for this option.

--tcp-flags [!] mask comp Match when the TCP flags are as specified. The first argument is the flags which we should examine, written as a comma-sepa rated list, and the second argument is a comma-separated list of flags which must be set. Flags are: SYN ACK FIN RST URG PSH ALL NONE. Hence the command iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST SYN will only match packets with the SYN flag set, and the ACK, FIN and RST flags unset.

[!] --syn Only match TCP packets with the SYN bit set and the ACK and RST bits cleared. Such packets are used to request TCP connection initiation; for example, blocking such packets coming in an interface will prevent incoming TCP connections, but outgoing TCP connections will be unaffected. It is equivalent to --tcp-flags SYN,RST,ACK SYN. If the "!" flag precedes the "--syn", the sense of the option is inverted.

The Firm's network with IPTABLES

The picture below shows The Firm's network and the possible combinations of traffic initiation/destination. We will go through six possible scenario's.



(1) Traffic initiated by the firewall and destined for the Internet

We are running a DNS on the Firewall that needs to be able to consult other DNS servers on the Internet (which use the UDP protocol and listen to port 53). We also 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 (RC5 64 bit 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 (the **ping** command uses the ICMP protocol and message type 8 (echo-request)). The Firewall communicates with the Internet through interface eth1. Taking all this into consideration, the **iptables** commands needed 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 echo-request      \
-m state --state NEW -j ACCEPT
```

These four **iptables** commands tell the firewall to allow outgoing connection-initialization packets for DNS, SSH, TCP/2064 and ping.

(2) Traffic initiated from the Internet and 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** command needed is:

```
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 and 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** command needed is:

```
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 and 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 needed 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 echo-request      \
-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 and destined for the Internet

Every connection from a machine on the internal network to a machine on the Internet is allowed. The **iptables** command needed is:

```
iptables -t filter -A FORWARD -i eth0 -o eth1 -m state --state NEW -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 101.102.103.104
```

This **iptables** command tells the firewall to allow ALL outgoing connection initialization packets.

(6) Traffic initiated by the Internet and 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 from 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 Internet 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 Internet that is aimed at port 80 should be routed to port 2345 on Machine 2. The **iptables** commands needed 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 this traffic will be allowed.

(!) 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 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 these last rules can't cause a security problem because the packets allowed 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 BUILT-IN CHAINS TO DROP
################################################################################
iptables -P INPUT   DROP
iptables -P FORWARD DROP
iptables -P OUTPUT  DROP

################################################################################
# (1) TRAFFIC INITIATED BY THE FIREWALL AND 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 echo-request      \
-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 FROM THE INTERNET AND 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 AND 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 AND 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 echo-request      \
-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 AND 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 FROM THE INTERNET AND 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 -o eth1 -j SNAT --to-source 101.102.103.104

################################################################################
# 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 **iptables-save** (which writes to `stdout`) and **iptables-restore** (which reads from `stdin`). Assuming that we use the file `fwrules.saved` to save and/or restore the rules, the two commands are:

```
iptables-save > fwrules.saved
iptables-restore < fwrules.saved
```

These commands can be used to initialize a firewall/routing machine at boottime by putting them into a SysV startup script.

Port and/or IP forwarding

In the fifth example of this chapter the client thinks it's connecting directly to the external server. The router in between transparently routes all traffic and performs SOURCE NAT to masquerade the internal (private) addresses. Keyword in this example is transparency. If you need NAT for outgoing traffic to the internet you can use SNAT (specifying the WAN IP address) if you have a static WAN IP address. The kernel's connection tracking keeps track of all the connections when the interface is taken down and brought back up. This is not the case when using MASQUERADE. Using MASQUERADE is a better idea when you have a dynamic WAN IP address because you don't have to specify the used IP address but can specify the used interface. Whatever IP address is on that interface, it is applied to al the outgoing packets.

Besides packet filtering, firewalls can also perform port and IP forwarding. In the sixth example (with port forwarding) an outside client will connect to a port on the firewall. To the client the connection will terminate on the firewall, but the firewall knows to what internal server connections on the receiving port should be forwarded. The firwewall will apply DESTINATION NAT to the packets and pass them on.

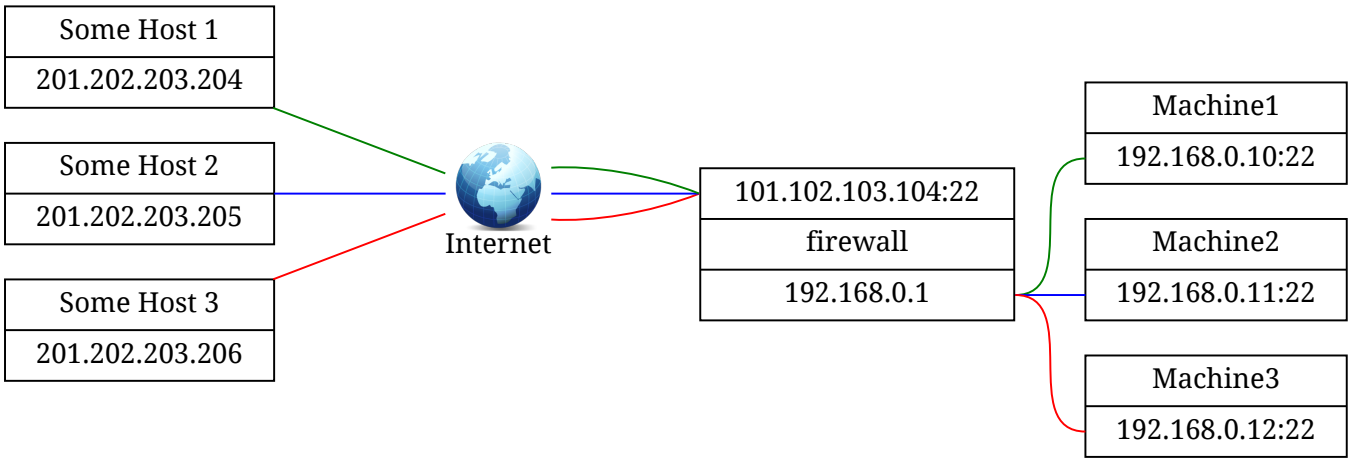
Forwarded traffic can also have SOURCE NAT applied but in most cases this is undesired because this would seriously hamper auditing the connections on the receiving server(s). There is no way of telling the original source address in that case - all traffic seems to originate at the firewall.

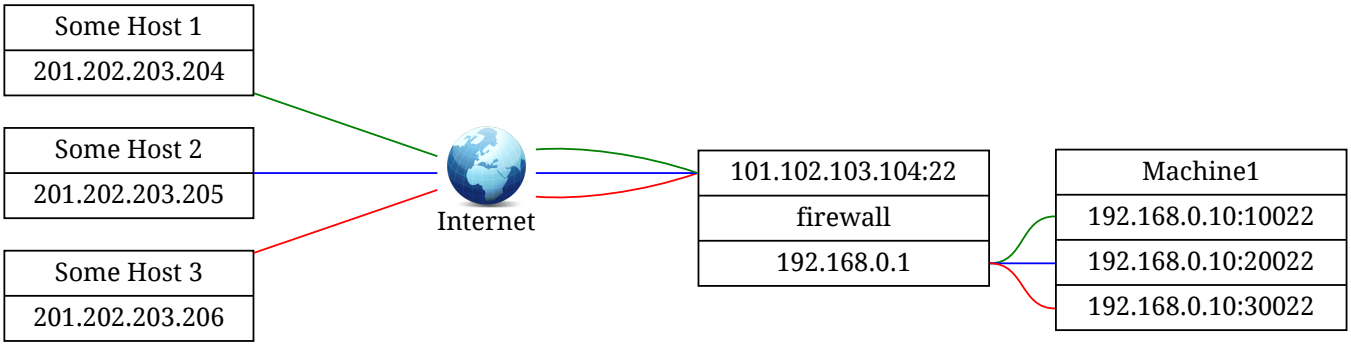
Situations in which port forwarding is used are (amongst others):

- Transparency is unwanted (security or other considerations):
  - Example: One specific service available through the firewall is running on multiple internal servers, for instance one for each customer of our organization. Based on the source address a decision can be made to which server the traffic has to be forwarded.
- Transparency is impossible (private IP addresses aren't routed on the internet):
  - Example 1: If an organisation has only one external public IP address, but several services that have to be accessible from the internet running on different servers, these services will need to be made available to the outside as seemingly originating from one single IP address.
  - Example 2: One internal server is serving a single service but listening on different port numbers (for instance to seperate between customer). Clients will connect to the IANA assigned service port and based on source IP address the firewall will forward the traffic to the assigned destination port for that source address.
- Scaling considerations:
  - Example: If an organisation has only one external public IP address but several services that have to be accessible from the internet running on different servers these services will need to be made available to the outside as seemingly originating from one single IP address.

Most often port and/or IP forwarding is used to enable incoming connections from the internet to servers with a private IP address.

Port forwarding examples:





Denial of Service (DoS) attacks

Description

DoS attackers abuse 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 often achieved 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 (character generator protocol) and echo UDP services are used.

Also, two systems (A and B) can be played out against each other by a third system (C). System C sends packets to system A but changes the source IP address of the packages it sends to the IP address of system B. System A then thinks the packets came from system B and starts sending replies to system B. 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.

Prevention

It is impossible to fully prevent DoS attacks without disconnecting from the Internet. What can be done to minimize the effects of DoS and DDoS attacks is to apply some packet filtering and rate limiting rules to the firewall.

Using /proc/sys/net/ipv4 (sysctl) to prevent simple DOS attacks

The kernel documentation describes the following sysctl options to prevent simple DoS attacks:

- tcp_max_orphans - INTEGER:
  - Maximal number of TCP sockets not attached to any user file handle, held by system. If this number is exceeded orphaned connections are reset immediately and a warning is printed.
- tcp_max_tw_buckets - INTEGER:
  - Maximal number of timewait sockets held by system simultaneously. If this number is exceeded time-wait socket is immediately destroyed and a warning is printed.
- rp_filter - INTEGER:
  - 0 - No source validation.
  - 1 - Strict mode as defined in RFC3704 Strict Reverse Path: Each incoming packet is tested against the FIB and if the interface is not the best reverse path the packet check will fail. By default failed packets are discarded.
  - 2 - Loose mode as defined in RFC3704 Loose Reverse Path: Each incoming packet's source address is also tested against the FIB and if the source address is not reachable via any interface the packet check will fail.

Routed

In an environment that uses dynamic routing, the *routed* daemon may be used. The routed daemon manages the routing tables in the kernel. The routed daemon only implements the RIP (Routing Information Protocol) protocol. When you wish to dynamically route other types of protocols gated can be used instead. Gated is a vintage routing daemon which supports RIPv2, RIPng, OSPF, OSPF6, BGP4+ and BGP4-.

The routed daemon finds interfaces to directly connected hosts and networks that are configured into the system and marked as up. (Mark networks as up using the ifconfig command.) If multiple interfaces are present, the routed daemon assumes that the local host forwards packets between networks. The routed daemon transmits a RIP request packet on each interface, using a broadcast message when the interface supports it.

The routed daemon then listens for RIP routing requests and response packets from other hosts. When the routed daemon supplies RIP information to other hosts, it sends RIP update packets every 30 seconds (containing copies of its routing tables) to all directly connected hosts and networks.

When the routed daemon receives a Routing Information Protocol (RIP) request packet to supply RIP routing information, the routed daemon generates a reply in the form of a response packet. The response packet is based on the information maintained in the kernel routing tables and contains a list of known routes. Each route is marked with a hop-count metric, which is the number of gateway hops between the source network and the destination network. The metric for each route is relative to the sending host. A metric of 16 or greater is considered infinite or beyond reach.

When to use routed

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 for some reason is unusable) the **routed** program can do this for you automatically.

Tools, commands and utilities to manage routing tables

route

route manipulates the kernel's IP routing tables. Its primary use is to set up static routes to specific hosts or networks via an interface after is has been configured with the **ifconfig** command. (The newest Linux distro's use **ip route** these days instead of **route**.)

SYNOPSIS (most important opions):

```
route
route [-v] add [-net|-host] target [netmask NM] [gw GW] [metric N] [[dev] If]
route [-v] del [-net|-host] target [netmask NM] [gw GW] [metric N] [[dev] If]
```

route itself, without any parameters, displays the routing table. The `-ee` option will generate a very long line with all paramaters from the routing table.

-v Select verbose operation.

-net|-host The target is a network or a host.

netmask Nm When adding a network route, the netmask to be used.

gw GW Route packets via a gateway. The specified gateway must be reachable first.

metric N Set the metric field in the routing table (used by routing daemons) to N.

dev If Force the route to be associated with the specified device. If *dev If* is the last option on the command line, the word *dev* may be omitted, as it's the default. Otherwise the order of the route modifiers (metric, netmask, gw, dev) doesn't matter.

Examples:

```
route add -net 192.168.10.0 netmask 255.255.255.0 dev eth0
route add -net 192.168.20.0 netmask 255.255.255.0 gw 192.168.1.10
route add default gw 192.168.1.1 eth1
```

The output of the kernel routing table is organized in the following columns:

Destination The destination network or destination host.

Gateway The gateway address or "*" if none is set.

Genmask The netmask for the destination net; "255.255.255.255" for a host destination and "0.0.0.0" for the *default* route.

Flags Possible flags include:

U - Route is up
H - Target is a host
G - Use gateway
R - Reinstate route for dynamic routing
D - Dynamically installed by daemon or redirect
M - Modified from routing daemon or redirect
C - Cache entry
! - Reject route

Metric The "distance" to the target (usually counted in hops).

Ref Number of references to this route.

Iface Interface to which packets for this route will be sent.

Example:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.20.0    *               255.255.255.0   U     0      0        0 eth0
link-local      *               255.255.0.0     U     1002   0        0 eth0
default         192.168.20.1    0.0.0.0         UG    0      0        0 eth0
```

netstat

Print network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.

netstat has a lot of possible options but the most used is the following:

```
# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.20.0    0.0.0.0         255.255.255.0   U         0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
0.0.0.0         192.168.20.1    0.0.0.0         UG        0 0          0 eth0
```

netstat -rn will show also the routing table. The `-r` option will show the routing table where the `-n` will prevent resolving IP addresses and networks to names. Please look at the man pages for more interesting options.

ip6tables

Ip6tables is the ipv6 equivalent of iptables. The syntax is identical to its ipv4 counterpart, except for the use of 128-bit addresses instead of 32-bit addresses.

The following example allows ICMPv6:

```
ip6tables -A INPUT -p icmpv6 -j ACCEPT
ip6tables -A OUTPUT -p icmpv6 -j ACCEPT
```

Iptables and ip6tables may be used simultanously. Refer to the `iptables(8)` manpage for detailed information.

Managing FTP servers (212.2)

Candidates should be able to configure an FTP server for anonymous downloads and uploads. This objective includes configuring user access, and precautions to be taken if anonymous uploads are permitted.

Key Knowledge Areas

Configuration files, tools and utilities for Pure-FTPd and vsftpd

Awareness of ProFTPd

Understanding of passive vs. active FTP connections

Terms and Utilities

- `vsftpd.conf`
- important Pure-FTPd command line options

FTP connection modes

FTP is a service that uses two ports for communication. Port 21 is used for the command port (also known as control port) and port 20 for the data port. FTP has two modes, *active* and *passive* FTP. These modes differ in the way connections are initiated. In active mode the client initates the control connection and the server initiates the data connection. In passive mode the client initiates both connections.

Active mode

In active mode the client starts an FTP session. This is done by initiating a control connection originating on an unprivileged port (>1023) to port 21 on the server. The client sends the server the IP address and port number on which the client will listen for the data connection. Usually this port is the next port above the used control connections port on the client. The server sends an ACK to the clients command port and *actively* opens a data connection originating on port 20 to the client. The client sends back an ACK on the data connection.

Active mode example:

- The client opens up a command channel from client port 1050 to server port 21.
- The client sends *PORT 1051* (1050 + 1) to the server and the server acknowledges on the command channel.
- The server opens up a data channel from server port 20 to client port 1051.
- The client acknowledges on the data channel.

Passive mode

In situations in which the client is behind a firewall and unable to accept incoming TCP connections, passive mode may be used. In passive mode the client starts an FTP session. This is done by initiating a control connection originating on an unprivileged port (>1023) to port 21 on the server. In this mode the client sends a PASV command to the server and receives an IP address and port number in return. The server replies with *PORT XXXX* where XXXX is the unprivileged port the server listens for the data connection and *passively* waits for the data connection. The client opens the data connection from the next port above the control connections port to the port specified in the **PORT** reply on the server. The server sends back an ACK to the client on the data connection.

Passive mode example:

- Client opens up command channel from client port 1050 to server port 21.
- Client sends *PASV* command to server on command channel.
- Server sends back (on command channel) *PORT 1234* after starting to listen on that port.
- Client opens up data channel from client 1050 to server port 1234.
- Server acknowledges on data channel.

Enabling connections through a firewall

To enable passive FTP connections when iptables is used, the “ip_conntrack_ftp” module has to be loaded into the firewall and connections with the state “related” have to be allowed.

vsftpd

vsftpd (very secure FTP daemon) is a very popular, versatile, fast and secure FTP server.

Example minimal configuration for anonymous up- and downloads

Example vsftpd.conf

```
# If enabled, vsftpd will run in standalone mode. This means  that
# vsftpd  must not be run from an inetd of some kind. Instead, the
# vsftpd executable is run once directly. vsftpd itself will  then
# take care of listening for and handling incoming connections.
# Default: NO
listen=NO

# Controls whether local logins are permitted or not. If enabled,
# normal user accounts in /etc/passwd (or wherever your PAM config
# references) may be used to log in. This must be enable for any
# non-anonymous login to work, including virtual users.
# Default: NO
local_enable=YES

# This controls whether any FTP commands which change the filesystem
# are  allowed  or not. These commands are: STOR, DELE, RNFR,
# RNTO, MKD, RMD, APPE and SITE.
# Default: NO
write_enable=YES

# Controls  whether  anonymous  logins  are  permitted  or not. If
# enabled, both the usernames ftp and anonymous are recognised  as
# anonymous logins.
# Default: YES
anonymous_enable=YES

# This option represents a directory  which  vsftpd  will  try  to
# change  into  after  an  anonymous  login.  Failure  is silently
# ignored.
# Default: (none)
anon_root=/var/ftp/pub

# If set to YES, anonymous users will be permitted to upload files
# under  certain  conditions.  For  this  to  work,   the   option
# write_enable  must be activated, and the anonymous ftp user must
# have write permission on desired upload locations. This  setting
```

```
# is also required for virtual users to upload; by default, virtual
# users are treated with anonymous (i.e. maximally restricted) privilege.
# Default: NO
anon_upload_enable=YES

# When enabled, anonymous users will only be allowed to download
# files which are world readable. This is recognising that the ftp
# user may own files, especially in the presence of uploads.
# Default: YES
anon_world_readable_only=NO
```

Create the ftp user:

```
useradd --home /var/ftp --shell /bin/false ftp
```

Create the FTP directory:

```
mkdir -p --mode 733 /var/ftp/pub/incoming
```

Set up inetd to listen for FTP traffic and start vsftpd. Add the following line to `/etc/inetd.conf`:

```
ftp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/vsftpd
```

Reload the inetd daemon.

An online HTML version of the manual page which lists all vsftpd config options can be found at: *Manpage of vsftpd.conf*.

When anonymous users should only be allowed to upload files, e.g., for sending files for analysis to remote support, make sure this directory is read-writable by the owner, root, and writeable but not readable by group members and others. This allows the anonymous user to write into the incoming directory but not to change it.

Pure-FTPd

Pure-FTPd is a highly flexible, secure and fast FTP server.

Configuration

Unlike many daemons, Pure-FTPd doesn't read any configuration file (except for LDAP and SQL when used). Instead, it uses command-line options. For convenience a wrapper is provided which reads a configuration file and starts Pure-FTPd with the right command-line options.

Specific configuration options of **pure-ftpd** can be found at: *Pure-FTPd Configuration file*.

Important command line options

If you want to listen for an incoming connection on a non-standard port, just append `-S` and the port number:

```
/usr/local/sbin/pure-ftpd -S 42
```

If your system has many IP addresses and you want the FTP server to be reachable on only one of these addresses, let's say 192.168.0.42, just use the following command:

```
/usr/local/sbin/pure-ftpd -S 192.168.0.42,21
```

Remark
The 21 port number could be left away since this is the default port.

To limit the number of simultaneous connections use the `-c` option:

```
/usr/local/sbin/pure-ftpd -c 50 &
```

Example minimal configuration for anonymous up- and downloads

Create the ftp user:

```
useradd --home /var/ftp --shell /bin/false ftp
```

Create the ftp directory structure with the correct permissions:

```
# Set the proper permissions to disable writing
mkdir -p --mode 555 /var/ftp
mkdir -p --mode 555 /var/ftp/pub
# Set the proper permissions to enable writing
mkdir -p --mode 755 /var/ftp/pub/incoming
```

Change ownership:

```
chown -R ftp:ftp /var/ftp/

192552    0 dr-xr-xr-x   3 ftp      ftp            16 Mar 11 11:54 /var/ftp
192588    0 dr-xr-xr-x   3 ftp      ftp             8 Mar 11 11:07 /var/ftp/pub
192589    0 drwxr-xr-x   2 ftp      ftp             8 Mar 11 11:55 /var/ftp/pub/incoming
```

Set up inetd to listen for FTP traffic and start **pure-ftpd**. Add the following line to `/etc/inetd.conf`:

```
ftp   stream   tcp   nowait   root   /usr/sbin/tcpd   /usr/sbin/pure-ftpd -e
```

Reload the inetd daemon:

```
killall -HUP inetd
```

or

```
kill -HUP $(cat /var/run/inetd.pid)
```

Other FTP servers

There are numerous FTP servers available and in use on Linux systems. Some alternatives to the servers mentioned above are: wu-ftpd and ProFTPd.

ProFTPd

ProFTPd - Professional configurable, secure file transfer protocol server.

```
SYNOPSIS
  proftpd [ -hlntv ] [ -c config-file ] [ -d debuglevel ] [ -p 0|1 ]
```

proftpd is the Professional File Transfer Protocol (FTP) server daemon. The server may be invoked by the Internet "super-server" **inetd(8)** each time a connection to the FTP service is made, or alternatively it can be run as a standalone daemon.

When **proftpd** is run in standalone mode and it receives a SIGHUP then it will reread its configuration file. When run in standalone mode without the -n option, the main **proftpd** daemon writes its process ID to `/var/run/run/proftpd.pid` to make it easy to know which process to SIGHUP.

See the man page of **proftpd** for detailed information on this ftp server. Detailed information can be found at: *The ProFTPd Project*.

Secure shell (SSH) (212.3)

Candidates should be able to configure and secure an SSH daemon. This objective includes managing keys and configuring SSH for users. Candidates should also be able to forward an application protocol over SSH and manage the SSH login.

Key Knowledge Areas

OpenSSH configuration files, tools and utilities

Login restrictions for the superuser and the normal users

Managing and using server and client keys to login with and without password

Usage of multiple connections from multiple hosts to guard against loss of connection to remote host following configuration changes

Terms and utilities

- **ssh**
- **sshd**
- `/etc/ssh/sshd_config`
- `/etc/ssh/`
- Private and public key files
- PermitRootLogin, PubkeyAuthentication, AllowUsers, PasswordAuthentication, Protocol

SSH client and server

ssh is a client program for logging into a remote machine and for executing commands on a remote machine.

sshd is the server (daemon) program for ssh.

Together these two programs replace **rlogin** and **rsh**, providing secure encrypted communications between two untrusted hosts on an insecure network.

To copy files over an insecure network the **scp** command can be used. Scp stands for Secure CoPy. It uses **ssh** for data transfer.

SSH (Secure SHell) uses digital keys for both data encryption and authentication.

Keys and their purpose

Two types of keys are used: *host* and *user* ones. These will be discussed in the next paragraphs.

Host keys

There is a slight difference between the SSH protocol versions 1 and 2 in socalled forward security.

SSH PROTOCOL VERSIONS:

SSH protocol version 1 This version only supports RSA keys. Each node has a *host* key (normally 2048 bits) to identify it. When the daemon is started, an additional *server* key (normally 768 bits) is generated. It is not stored on disk and recreated every hour when used.

When a client connects, the server 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 and server keys, and sends the encrypted number to the server. Both sides then use this random number as a key that 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 from those offered by the server.

SSH protocol version 2 This protocol version is the default and it supports DSA, ECDSA and RSA keys. 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-md5, hmac-sha1, umac-64, umac-128, hmac-ripemd160, hmac-sha2-256 or hmac-sha2-512).

Finally, in both versions of the protocol, the server and the client enter an authentication dialog. The client tries to authenticate itself using host-based authentication, public key authentication, challange-response authentication, or password authentication. If possible, select SSH protocol version 2 as the only one to use.

User keys, public and private

ssh implements the RSA authentication mechanism automatically. The user creates an RSA key pair by running **ssh-keygen**. This stores the private key in `$HOME/.ssh/id_rsa` and the public key in `$HOME/.ssh/id_rsa.pub` in the user's home directory. The user should then copy the id_rsa.pub into the `$HOME/.ssh/authorized_keys` file in his home directory on the remote machine:

```
# cat id_rsa.pub >> ~/.ssh/authorized_keys
```

The `authorized_keys file` has only one key per line which can be very long. After this, the user can log in without giving the password. Instead of using rsa, dsa can also be used. The names of the keys reflect the kind of keys you created. Make sure the ~/.ssh directory and the files in it have the proper rights. Use for example 700 on the .ssh directory and 600 on the files in it. If you don't, in some situations you can't login using digital keys.

Configuring sshd

Configuring **sshd** can be done by using command-line options or by editing the configuration file `/etc/ssh/sshd_config`.

-4 Forces sshd to use IPv4 addresses only.

-6 Forces sshd to use IPv6 addresses only.

-b ***bits*** Specifies the number of bits in the ephemeral protocol version 1 server key (default 1024).

-C ***connection_spec*** Specify the connection parameters to use for the -T extended test mode.

-D When this option is specified, **sshd** will not detach and does not become a daemon. This allows easy monitoring of sshd.

-d ***Debug Mode*** The server sends verbose debug output to the system log, and does not put itself in the background.

-e When this option is specified, sshd will send the output to the standard error instead of the system log.

-f ***config_file*** Specifies the name of the configuration file. The default is /etc/ssh/sshd_config. sshd refuses to start if there is no configuration file.

-g ***login_grace_time*** Gives the grace time for clients to authenticate themselves (default 120 seconds).

-h ***host_key_file*** Specifies a file from which a host key is read.

-i Specifies that sshd is being run from **inetd**. sshd is normally not run from inetd because it needs to generate the server key before it can respond to the client, and this may take tens of seconds.

-k ***key_gen_time*** Specifies how often the ephemeral protocol version 1 server key is regenerated (default 3600 seconds, or one hour).

-o ***option*** Can be used to give options in the format used in the configuration file. This is useful for specifying options for which there is no separate command-line flag.

-p ***port*** Specifies the port on which the server listens for connections (default 22). Multiple port options are permitted.

-q Quiet mode. Nothing is sent to the system log. Normally the beginning, authentication, and termination of each connection is logged.

-T Extended test mode. Check the validity of the configuration file, output the effective configuration to stdout and then exit.

-t Test mode. Only check the validity of the configuration file and sanity of the keys. This is useful for updating sshd reliably as configuration options may change.

-u ***len*** This option is used to specify the size of the field in the utmp structure that holds the remote host name.

The sshd configrations file in the `/etc/ssh/sshd_config` directory can also be used to configure sshd. This file should be writable by root only, but it is recommended (though not necessary) that it is world-readable.

Allow or deny root logins

The allowing or denying of root logins 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. Without the possibility of remote root access someone who wants to get root access on a remote server has to get access via a regular user first. Which is an additional layer of security.

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 with public key authentication and execute the commands that are given on the command line. The allowed command(s) must be added to the public key(s) in the `~/.ssh/authorized_keys` file. For example, if you add *command="/bin/date"* at the beginning of your public key line on SERVERA, only the date command will be allowed for a login using that public key. If you execute **ssh root@SERVERA** then only the date command will be executed on the remote system. (This may be useful for taking remote backups even if root login is normally not allowed.) Please read the man pages for more information.

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.

SSHD KEYWORDS:

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 cannot 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.

PasswordAuthentication Specifies whether password authentication is allowed. The default is "yes".

Protocol Specifies the protocol versions sshd supports. The possible values are "1" and "2". Multiple versions must be comma-separated. The default is "2,1". Note that the order of the protocol list does not indicate preference, because the client selects among multiple protocol versions offered by the server. Specifying "2,1" is identical to "1,2". Unless there are serious arguments for using protocol version "1", only use version "2" because it is far more secure.

UsePAM Enables the Pluggable Authentication Modules interface. If set to “yes” it will enable PAM authentication using *ChallengeResponseAuthentication* and *PasswordAuthentication* in addition to PAM account and session module processing for all authentication types. The default is “yes”.

ChallengeResponseAuthentication Specifies whether challenge-response authentication is allowed (e.g., via PAM or through authentication styles supported in login.conf(5)). The default is “yes”.

If you want to fully disable password based logins, the following `sshd_config` settings should be set to no:

- PasswordAuthentication
- ChallengeResponseAuthentication
- UsePAM

Enabling or disabling X forwarding

This topic stays here to give you some information about this topic but it isn’t anymore in the Key Knowledge Areas of lpi.org.

There are a number of keywords that can be used to influence the behaviour of **sshd** in relation to the X Windows system.

X WINDOWS KEYWORDS (SSHD_CONFIG):

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 `DISPLAY` in the server-shell to **localhost:num:0** which is actually the tunnel-endpoint which is mapped back to the original `DISPLAY` 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 be transferred through the SSH tunnel to machine A. To display the terminal, machine B will connect to the display on localhost that’s opened by SSH for X11 Forwarding. SSH will forward the X-connection to X server on machine A where it will be displayed in your local display. Because the X output is seemingly created locally no xhost settings have to be changed to enable displaying the content.

To enable X11 forwarding this also has to be enabled on the client side. ForwardX11 has to be set to "yes" in the SSH configuration on the client, or on the command line when initiating the connection.

Passwordless authentication

Using SSH there are different ways for authentication. One way to do this is by using a public/private keypair for authentication. If public key authentication is enabled users can login without having to use a password. When trying to authenticate with a keypair the user will be asked for the passphrase (or ssh_agent will be queried). Also, passphraseless keys can be used (e.g., for automated sessions).

Note on using passphraseless keys: because no passphrase is needed to use a passphraseless key, anyone who has access to the key can use it. This poses a serious security risk. The impact of connections with passphraseless keys should be reduced by allowing these connections only with forced commands and preferably from a limited set of clients, (see man authorized_keys).

PASSWORDLESS AUTHENTICATION KEYWORD:

PubkeyAuthentication This parameter specifies whether public key authentication is allowed. The default is "yes". Note that this option applies to protocol version 2 only.

ssh-agent

This objective was moved to LPIC-1.

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 the use of environment variables the agent can be located and automatically used when logging in to other machines with **ssh**.

Enable agent forwarding

In `ssh_config` (system wide or in `$HOME/.ssh/config`) set *ForwardAgent* to "yes".

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_rsa` (or id_dsa) which contains the private key, as described earlier. You will be prompted to enter your passphrase, if necessary.

When the user logs out from the 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.

Enabling X-sessions with ssh-agent

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.

Tunneling an application protocol over ssh with portmapping

Description

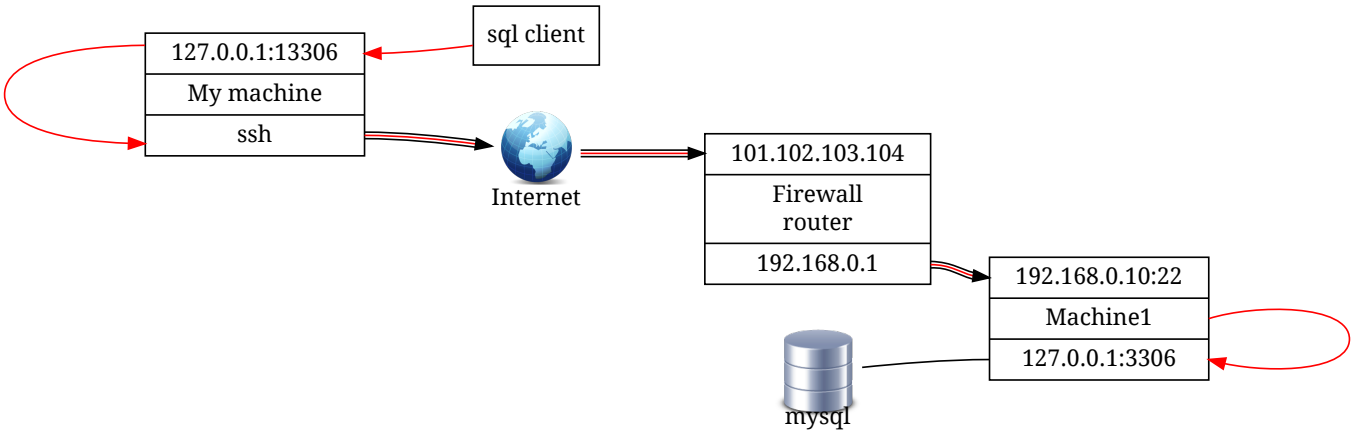
Using port forwarding, SSH will bind to a port and tunnel all traffic directed at that port through the SSH connection. Traffic is forwarded to the host at the other end of the SSH connection. The remote host may be the server on which the SSH connection terminates, but traffic can also be forwarded to another host. Port forwarding can be configured in both directions: local to remote, but also remote to local.

The syntax is:

```
ssh -R|L [bind_address:]port:host:host_port [user@]hostname [command]
```

Example

As an example (example 1), consider the situation shown in the picture. We are working on MyMachine, and we want to connect to the mysql server on Machine 1. The firewall doesn't allow sql connections. The firewall has port forwarding configured so that incoming traffic on port 22 from the internet will be forwarded to port 22 on Machine1. John is an user on Machine1.



First open the tunnel by running **ssh** with the "-L" option:

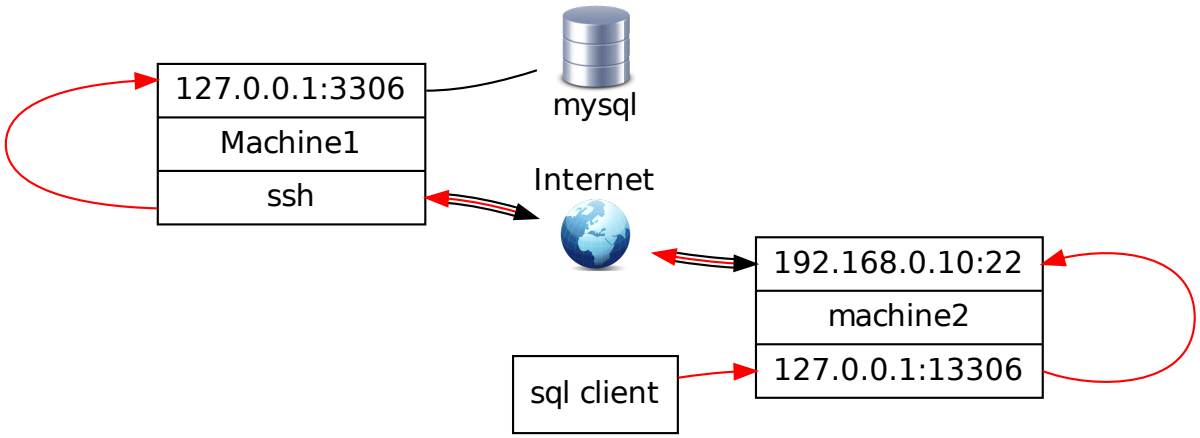
```
MyMachine# ssh -L 13306:localhost:3306 john@101.102.103.104
```

Then run your application, connecting to the local port that is forwarded by the SSH tunnel:

```
MyMachine# mysql -P 13306
```

You are now connected to the remote MySQL server without the need to enable SQL connections over the network.

Another example (example 2) would be where one side of the connection wants/needs to initialize the SSH connection to enable the remote party to connect back through the tunnel. In this simplified example an ssh tunnel is started on Machine1 to machine2 with the `-R`. Then the sql client on machine2 can connect on localhost:13306 and gets a connection with the mysql server on port 3306 through the SSH tunnel.



The tunnel will stay open as long as the SSH session is running or as long as there is a tunneled session going through it. This means there are two ways of working with SSH tunneling:

- open an SSH session with port forwarding and keep it open for as long as you need it. This is useful if you want to be able to reconnect through the tunnel without the need to open a new SSH session. Useful for connecting to a remote host and initializing a tunnel in the reversed direction (example 2).
- open an SSH connection with port forwarding in the background and keep it open just long enough to be able to initialize the connection through the tunnel. ("ssh -f -L <your:port:definition> remote_user@remote_host sleep 10 ; <your application>").

Security tasks (212.4)

Candidates should be able to receive security alerts from various sources, install, configure and run intrusion detection systems and apply security patches and bugfixes.

Key Knowledge Areas:

Tools and utilities to scan and test ports on a server

Locations and organizations that report security alerts as Bugtraq, CERT, or other sources

Tools and utilities to implement an intrusion detection system (IDS)

Awareness of OpenVAS and Snort

Terms and utilities:

- **telnet**
- **nmap**
- **fail2ban**
- **nc**
- **OpenVAS**
- **Snort IDS**

nc (netcat)

Description

Netcat (nc) is a very versatile network tool. Netcat is a computer networking service for reading from and writing to network connections using TCP or UDP. Netcat is designed to be a dependable "back-end" device that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and investigation tool. Netcat's features are numerous; Netcat can, for instance, be used as a proxy or portforwarder. It can use any local source port, or use loose source-routing. It is commonly referred to as the TCP/IP Swiss army knife.

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**.

Example netcat. Using netcat to perform a port scan

With the -z option netcat will perform a portscan on the ports given on the command line. By default netcat will produce no output. When scanning only one port the exit status indicates the result of the scan, but with multiple ports the exit status will allways be "0" if one of the ports is listening. For this reason using the "verbose" option will be useful to see the actual results:

```
# nc -vz localhost 75-85
nc: connect to localhost port 75 (tcp) failed: Connection refused
nc: connect to localhost port 76 (tcp) failed: Connection refused
nc: connect to localhost port 77 (tcp) failed: Connection refused
nc: connect to localhost port 78 (tcp) failed: Connection refused
Connection to localhost 79 port [tcp/finger] succeeded!
Connection to localhost 80 port [tcp/http] succeeded!
nc: connect to localhost port 81 (tcp) failed: Connection refused
nc: connect to localhost port 82 (tcp) failed: Connection refused
nc: connect to localhost port 83 (tcp) failed: Connection refused
nc: connect to localhost port 84 (tcp) failed: Connection refused
nc: connect to localhost port 85 (tcp) failed: Connection refused
```

The man page of netcat shows some more examples on how to use netcat.

Netcat can easily be used in scripts for a lot of tests you want to run automated.

The fail2ban command

Description

Fail2ban scans log files like `/var/log/pwdfail` or `/var/log/apache/error_log`, and bans IP addresses that cause too many rejected password attempts. It updates firewall rules to block the IP addresses.

Fail2ban's main function is to block IP addresses that belong to hosts that may be trying to breach the system's security. It determines these by monitoring log files (e.g. `/var/log/pwdfail`, `/var/log/auth.log`, etc.) and bans any host IP that does too many login attempts or performs any other unwanted action within a time frame set by the administrator. Fail2ban is typically configured to unban a blocked host after a certain period, so as to not "lock out" any genuine connections. An unban time of several minutes is usually sufficient to prevent a network connection from being flooded by malicious attempts, as well as to reduce the likelihood of a successful dictionary attack.

The nmap command

Description

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 wish to check that no ports are open that you do not want open, **nmap** is the tool to use.

Using the nmap command

If a machine gets infected by a rootkit, some system utilities like **top**, **ps** and **netstat** will usually be replaced by the attacker. The modified versions of these commands aide the attacker by not showing all available processes and listening ports. By performing portscans against our host we can explore which ports are open, and compare this with a list of known services. As an example, here's an example of a TCP portscan against our localhost:

```
$ nmap -sT localhost

Starting Nmap 6.25 ( http://nmap.org ) at 2013-07-04 06:33 CDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0011s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 993 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
389/tcp  open  ldap
3000/tcp open  ppp

Nmap done: 1 IP address (1 host up) scanned in 0.20 seconds
```

Note
By default, **nmap** will only scan the 1000 most common ports. Use the `-p 1-65535` or `-p -` switch to scan all available ports.

Let's perform the same scan, using the UDP protocol:

```
$ nmap -sU localhost
You requested a scan type which requires root privileges.
QUITTING!
```

Nmap is a very powerful network scanner, but some options require root privileges. If you would perform the command **nmap localhost** both as root and using your own privileges, nmap would use the `-sS` option as root and the `-sT` when run with normal user privileges.

Now, let's run the UDP scan again using root privileges trough sudo:

```
$ sudo nmap -sU localhost
[sudo] password for bob: ********

Starting Nmap 6.25 ( http://nmap.org ) at 2013-07-04 06:51 CDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000040s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 995 closed ports
PORT     STATE         SERVICE
53/udp   open          domain
68/udp   open|filtered dhcpc
111/udp  open          rpcbind
```

```
1900/udp open|filtered upnp
5353/udp open|filtered zeroconf

Nmap done: 1 IP address (1 host up) scanned in 1.48 seconds
```

Nmap is a very versatile and powerful tool, and offers a variety of options regarding its capabilities. Nmap can, for example, be used for active TCP/IP stack fingerprinting to determine the remote OS. You need administrator rights to do this:

```
[root@lnx1 ~]# nmap -A 192.168.1.183

Starting Nmap 5.51 ( http://nmap.org ) at 2015-10-30 16:03 CET
Nmap scan report for 192.168.1.183
Host is up (0.0012s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
| ssh-hostkey: 1024 65:44:cf:c2:b8:e9:6a:a5:21:18:9f:55:70:1d:d9:57 (DSA)
|_2048 87:15:36:b4:28:09:3c:84:fa:ea:9f:b3:9d:33:39:f9 (RSA)
MAC Address: 00:0F:60:02:BA:0D (Lifetron Co.)
No exact OS matches for host (If you know what OS is running on it, see http://nmap.org/ ↩
    submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=5.51%D=10/30%OT=22%CT=1%CU=34692%PV=Y%DS=1%DC=D%G=Y%M=000F60%TM=5
OS:63386C7%P=x86_64-redhat-linux-gnu)SEQ(SP=109%GCD=1%ISR=103%TI=Z%CI=I%II=
OS:I%TS=7)SEQ(SP=107%GCD=1%ISR=104%TI=Z%CI=I%II=I%TS=7)OPS(O1=M5B4ST11NW6%O
OS:2=M5B4ST11NW6%O3=M5B4NNT11NW6%O4=M5B4ST11NW6%O5=M5B4ST11NW6%O6=M5B4ST11)
OS:OPS(O1=M5B4ST11NW6%O2=NNT11%O3=M5B4NNT11NW6%O4=M5B4ST11NW6%O5=M5B4ST11NW
OS:6%O6=M5B4ST11)WIN(W1=7120%W2=7120%W3=7120%W4=7120%W5=7120%W6=7120)ECN(R=
OS:Y%DF=Y%T=40%W=7210%O=M5B4NNSNW6%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%F=AS%R
OS:D=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(R=Y%
OS:DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%
OS:O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=4
OS:0%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40%CD=S)

Network Distance: 1 hop
Service Info: OS: Linux

TRACEROUTE
HOP RTT     ADDRESS
1   1.16 ms 192.168.1.183
```

As you can tell from the output, the tested machine was a Debian linux host.

Please consult the manpage of `nmap(1)` to learn more about its features.

OpenVAS

The Open Vulnerability Assessment System (OpenVAS) is an open source framework of several services and tools offering a comprehensive and powerful vulnerability scanning and vulnerability management solution.

The actual security scanner is accompanied with a daily updated feed of Network Vulnerability Tests (NVTs), over 30,000 in total (as of April 2013).

Detailed information about OpenVAS can be found at: *Openvas - Open vulnerability assessment system community site* .

The Snort IDS (Intrusion Detection System)

Snort is an open source network intrusion detection system (NIDS) 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.

Basic structure of Snort rules

All Snort rules have two logical parts: rule *header* and rule *options*.

The rule header contains information about what action a rule takes. It also contains criteria for matching a rule against data packets. The options part usually contains an alert message and information about which part of the packet should be used to generate the alert message. The options part contains additional criteria for matching a rule against data packets. A rule may detect one type or multiple types of intrusion activity. Intelligent rules should be able to apply to multiple intrusion signatures.

Structure of Snort rule headers

The action part of the rule determines the type of action taken when criteria are met and a rule is exactly matched against a data packet. Typical actions are generating an alert or log message or invoking another rule. You will learn more about actions later in this chapter.

The protocol part is used to apply the rule on packets for a particular protocol only. This is the first criterion mentioned in the rule. Some examples of protocols used are IP, ICMP, UDP etc.

The address parts define source and destination addresses. Addresses may be a single host, multiple hosts or network addresses. You can also use these parts to exclude some addresses from a complete network. More about addresses will be discussed later. Note that there are two address fields in the rule. Source and destination addresses are determined based on direction field. As an example, if the direction field is "->", the Address on the left side is source and the Address on the right side is destination.

In case of TCP or UDP protocol, the port parts determine the source and destination ports of a packet on which the rule is applied. In case of network layer protocols like IP and ICMP, port numbers have no significance.

The direction part of the rule actually determines which address and port number is used as source and which as destination.

Just some examples:

```
alert icmp any any -> any any (msg: "Ping with TTL=100";  ttl: 100;)
alert udp any 1024:2048 -> any any (msg: "UDP ports";)
alert tcp 192.168.2.0/24 23 <> any any (content: "confidential"; msg: "Detected  ↩
    confidential";)
log udp any !53 -> any any log udp
```

Detailed information about Snort can be found at: *Snort IDS* .

Intrusion Detection and Prevention Systems

When talking about Intrusion Detection Systems (IDS), we can make a distinction between Host Intrusion Detection Systems (HIDS) and Network Intrusion Detection Systems (NIDS). A HIDS alerts when a host is suffering from suspicious activities. A NIDS usually inspects network traffic, preferably at a low level and alerts if suspicious traffic is detected.

Some IDS systems can be configured in a way that they do not only send out an alert, but also prevent access to a certain resource. This resource can either be a TCP/IP or UDP port, a physical port on a network device or complete access to a certain host or network segment trough a router or firewall. Since these systems not only detect, but also prevent they are called Intrusion Prevention Systems (IPS). As well as with IDS systems, we can distinguish HIPS from NIPS systems.

Both intrusion detection and intrusion prevention systems use a system of definitions for detection. These definitions describe certain characteristics that when met, trigger off an alert or countermeasure. If a detection takes place and is correct, we call this a *true positive*. If a detection takes place but is inaccurate, this is called a *false positive.*. When the system does not detect

something that does not occur, this is called a true negative. When there actually is an event which is not detected by the system, this is called a false negative.

Often, the detection capabilities of the IDS are expanded by using heuristic detection methods. In order for these to be both effective and accurate, the system needs to be trained. During this period, a lot of false positives may be detected which isn't a bad thing. But the system needs to be tweaked so the amount of false positives will be reduced to a minimum. A false negative is equal to having no IDS in place, and is the most undesirable behavior for an IDS.

Keeping track of security alerts

Security alerts

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

Description

BugTraq is a full disclosure moderated mailing-list at *securityfocus.com* for detailed discussion and announcement of computer security vulnerabilities: what they are, how to exploit them and how to fix them.

Bugtraq website

The *SecurityFocus* website brings together many different resources related to security. One of them is the Bugtraq mailing list. There also is a Bugtraq FAQ.

How to subscribe to Bugtraq

Use the webform at http://www.securityfocus.com/ to subscribe to any of the SecurityFocus mailing lists.

CERT

Description

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.

Website

CERT maintains a website called *The CERT Coordination Center*

How to subscribe to the CERT Advisory mailing list

See the us-cert.gov *lists and feed page* to sign up for the CERT Advisory mailing list or the RSS feeds issued on diverse NCAS publications.

CIAC

Description

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.

Website

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 CIAC mailing list, send the following request via E-mail to ciac-listproc@llnl.gov: unsubscribe list-name.

Testing for open mail relays with telnet

Description

An open mail relay is a mail server that accepts SMTP connections from anywhere and will forward emails to any domain. This means that everyone can connect to port 25 on that mail server and send mail to whomever they want. As a result your server's IP might end up on anti-spam blacklists.

Testing for open mail relaying

Testing a mail relay can be done by delivering an email for a recipient to a server that's not supposed to do any relaying for the recipients domain. If the server accepts AND delivers the email it is an open relay.

In the following example we use **telnet** to connect to a SMTP server running on port 25:

```
$ telnet localhost 25
Trying ::1...
Connected to localhost.
Escape character is '^]'.
220 linux.mailserver ESMTP Exim 4.80 Wed, 03 Jul 2013 08:08:06 -0500
MAIL FROM: bob@example.com
250 OK
RCPT TO: root@localhost
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
Open Mail Relay test message
.
250 OK id=1UuMnI-0001SM-Pe
QUIT
221 linux.mailserver closing connection
Connection closed by foreign host.
```

The message is accepted because the mailserver is configured to accept connections that origin from the local host, and because `root@localhost` is a valid email address according to the SMTP server.

Telnet is not considered very suitable as a remote login protocol because all data is being transmitted in clear text across the network. But the **telnet** command is very useful for checking open ports. The target port can be given as an argument, as can be seen in the example above.

OpenVPN (212.5)

Candidates should be able to configure a VPN (Virtual Private Network) and create secure point-to-point or site-to-site connections.

References: OpenVPN

Key Knowledge Areas:

OpenVPN

Terms and Utilities:

- `/etc/openvpn/`
- **openvpn**

OpenVPN

OpenVPN is a free and open source software application that implements virtual private network (VPN) techniques for creating secure point-to-point or site-to-site connections in routed or bridged configurations and remote access facilities. It uses SSL/TLS security for encryption and is capable of traversing network address translators (NATs) and firewalls.

OpenVPN allows peers to authenticate each other using a pre-shared secret key, certificates, or username/password. When used in a multiclient-server configuration, it allows the server to release an authentication certificate for every client, using signature and Certificate authority. It uses the OpenSSL encryption library extensively, as well as the SSLv3/TLSv1 protocol, and contains many security and control features.

Installing

OpenVPN is available on almost any modern operating system and can be built from source or installed as a pre-built package.

OpenVPN is not compatible with IPsec or any other VPN package. The entire package consists of one binary for both client and server connections, an optional configuration file, and one or more key files depending on the authentication method used.

openvpn options

OpenVPN allows any option to be placed either on the command line or in a configuration file. Though all command line options are preceded by a double-leading-dash ("--"), this prefix can be removed when an option is placed in a configuration file.

--config file Load additional config options from file where each line corresponds to one command line option, but with the leading "--" removed.

-dev tunX|tapX|null TUN/TAP virtual network device (X can be omitted for a dynamic device.).

---nobind *bits* Do not bind to local address and port. The IP stack will allocate a dynamic port for returning packets. Since the value of the dynamic port could not be known in advance by a peer, this option is only suitable for peers which will be initiating connections by using the --remote option.

--ifconfig l rn *connection_spec* Set TUN/TAP parameters. l is the IP address of the local VPN endpoint. For TUN devices, rn is the IP address of the remote VPN endpoint. For TAP devices, rn is the subnet mask of the virtual ethernet segment which is being created or connected to.

secret file [direction] Enable Static Key encryption mode (non-TLS). Use pre-shared secret file which was generated with --genkey

Configuration

Simple point-to-point example

This example uses static keys for authentication. This is a very simple setup, ideal for point-to-point networking. In the following example the tun interfaces will be used. Another possibility would be to use the tap interfaces but then the configuration would also be a little bit different. See the man pages for more information about using these interfaces.

A VPN tunnel will be created with a server endpoint of 10.10.10.10 and a client endpoint of 10.10.10.11. The public ipaddress of the server is referenced by vpnserver.example.com. The communication between these endpoints will be encrypted and occur over the default OpenVPN port 1194.

To setup this example a key has to be created: **openvpn --genkey --secret static.key**. Copy this key (`static.key`) to both client and server.

Server configuration file (server.conf):

```
dev tun
ifconfig 10.10.10.10 10.10.10.11
keepalive 10 60
ping-timer-rem
persist-tun
persist-key
secret static.key
```

Client configuration file (client.conf):

```
remote vpnserver.example.com
dev tun
ifconfig 10.10.10.11 10.10.10.10
keepalive 10 60
ping-timer-rem
persist-tun
persist-key
secret static.key
```

Start the vpn on the server by running **openvpn server.conf** and running **openvpn client.conf** on the client.

Questions and answers

System Security

1. *List the private network address ranges defined by IANA*

   - 10.0.0.0 - 10.255.255.255 (10.0.0.0/8)
   - 172.16.0.0 - 172.31.255.255 (172.16.0.0/12)
   - 192.168.0.0 - 192.168.255.255 (192.168.0.0/16)

   Private Network Addresses [339]

2. *What does the accronym NAT stand for?*

   Network Address Translation. Network Addresses Translation [340]

3. *How can a server with a private IP address connect to a server on the internet?*

   By connecting through a router (or server with router functionality) that performs Network Address Translation. Network Addresses Translation implementation [340]

4. *Which tool is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel?*

   iptables iptables [340]

5. *Name the netfilter CHAINS*

   PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING iptables chains

6. *Name the netfilter TABLES.*

   Filter, Nat, Mangle iptables tables

7. *What module is needed to perform stateful firewalling on FTP traffic?*

   ip_conntrack_ftp iptables ip_conntrack_ftp [342]

8. *What is needed for FTP to work through a firewall ("incoming")?*

   - Module ip_conntrack_ftp has to be loaded
   - Incoming NEW connections to port 21 have to be ACCEPTED
   - Incoming traffic for RELATED and ESTABLISHED connections have to be ACCEPTED
   - Outgoing traffic has to be accepted (minimal from port 21 and ESTABLISHED and RELATED).

   FTP through the firewall [357]

9. *Name the connection states a packet can be in when arriving at a stateful firewall.*

   NEW, ESTABLISHED, RELATED, INVALID iptables connection states [342]

10. *What are the minimal iptables rules to enable responding to a ping originating from any server on the network?*

    Assuming that the network is attached to eth0:

    ```
    iptables -t filter -A INPUT -i eth0 -p icmp --icmp-type echo-request -m state -j ACCEPT
    iptables -t filter -A OUTPUT -o eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
    ```

    iptables allow ICMP [349]

11. *What's the difference between FTP in active and passive mode?*

    In active mode, the client sends the server the IP address and port number on which the client will listen, and the server initiates the TCP connection. In passive mode the client sends a PASV command to the server and receives an IP address and port number in return. The client uses these to open the data connection to the server. FTP passive active [356]

12. *What protocol is implemented by the routed daemon?*

    The RIP routing protocol. routed implements RIP [354]

13. *Which 4 targets does iptables know by default?*

    ACCEPT, DROP, QUEUE, RETURN iptables default targets

14. *Name at least three extended iptables targets*

    LOG, MARK, REJECT, TOS, MIRROR, SNAT, DNAT, MASQUERADE, REDIRECT iptables extended targets

15. *Which modules are included in iptables by default? (name 4)*

    tcp, udp, icmp, mac, limit, multiport, mark, owner, state, unclean, tos iptables modules

16. *Describe "DoS with IP address spoofing"*

    System A sends packets to system B. These packets have the forged source address of system C. As a result system B will send responses to system C. DoS with IP address spoofing [353]

17. *How can DoS attacks be prevented?*

    DOS attacks cannot be prevented, but the impact can be reduced by applying filtering and rate limiting rules to the firewall. Preventing DoS [353]

18. *What is SSH?*

    SSH is a secure replacement for rlogin and rsh. SSH usage [360]

19. *Name the possible values for the sshd configuration option **PermitRootLogin***

    yes, no, without-password, forced-commands-only SSH PermitRootLogin [362]

20. *What is the prefered way to display x content from within an ssh session?*

    Enable X11 forwarding to foward X data to the local display over the SSH connection. SSH enable X forward

21. *Name a security implication of using passphrase ssh keys and at least one way to reduce the impact.*

    Anyone with access to the passphraseless key has access because no passphrase is needed to use the key. Set a forced command for the key and preferably limit access to one (or a few) client host(s). Passphraseless keys risk [363]

22. *How can you make sure you don't have to type in your passphrase for each new connection (using the same key)?*

    Use ssh-agent to load the key(s) and enable agent forwarding. ssh-agent [364]

23. *What is SNORT?*

    Snort is a network Intrusion Detection System. Section 12.4.7

24. *How can services on an internal servers with a private IP address be made available for access from the internet?*

    Configure port forwarding for incoming connections on the firewall. iptables port forwarding [352]

25. *What has to be done to enable passwordless logon?*

    Create a public/private key pair and add the contents of the public key to the authorized_keys file of the remote user. SSH authorized_keys [361]

26. *What is the most important reason not to perform SOURCE NAT on incoming connections from the internet?*

    It hampers auditing of these connections on the receiving server because all traffic will seem to originate from the same client (the firewall). iptables Source Nat consideration [352]

27. *How is running pure-ftpd different from running any other FTP server?*

    Unlike many daemons, Pure-FTPd doesn't read any configuration file (except for LDAP and SQL when used). Instead, it uses command-line options. Pure-FTPD configuration [358]

28. *Do you need to run xhost to allow connections to the local X server if we want to display X output generated in an SSH session with X11 forwarding enabled?*

    No. Because the X output is seemingly created locally no xhost settings have to be changed to enable displaying the content. SSH X11 forwarding without xhost [363]

29. *How does port forwarding with SSH work?*

    SSH binds a local port, tunnels all traffic from that port through the open SSH connection associated with the bound local port to a port on a server on the other side of that connection. SSH port mapping [364]

30. *What's the use of **nmap**?*

    Nmap can be used to scan a network to determine which hosts are up and what services they are offering. nmap [367]

31. *Describe openVAS.*

    OpenVAS is a framework of several services and tools offering a comprehensive and powerful vulnerability scanning and vulnerability management openvas [369]

32. *Name a few sources for security alerts.*

    - Bugtraq
    - CERT
    - CIAC

    Security alerts

Chapter 13

Bibliography

[Albitz01] Paul AlbitzCricket Liu, *DNS and BIND (Fourth Edition)* , O'Reilly, 2001, ISBN 0-596-00158-4.

[Apache01] *Apache HTTP Server Directive Index* , The Apache Software Foundation, 2013.

[Aplus901902] Mike Meyers, *CompTIA A+ Certification ALL in One Exam Guide, Ninth Edition* , McGraw Hill, 2016, ISBN-13 978-1-25-958951-5.

[BIND at Wikipedia.org] *BIND at Wikipedia.org*, Wikipedia.

[BINDfaq] , Internet Systems Consortium.

[Bandel97] David Bandel, Linux Journal, January 1997.

[Bar00] Moshe Bar, Dr Dobbs, March 29th, 2000.

[Bird01] Tina Bird, Aug 2001.

[BootFAQ]

[Brockmeier01] Joe Brockmeier, Unix review.

[Btrfs]

[Choose the right DNS configuration] *A Comparison of DNS Servers: How to choose the right DNS configuration*, Justin Ellingwood.

[ChrootBind9] Scott Wunsch, The Linux Documentation Project.

[Coar00] Ken Coar, INT Media Group, Incorporated, February 2000.

[Colligan00] Unknown, Wikipedia, November 2, 2012.

[DANE at Wikipedia.org] *DANE at Wikipedia.org*, Wikipedia.

[DNShowto] Nicolai LangfeldtJamie Norrish, v3.1: October 2001.

[Dawson00] Terry DawsonOlaf Kirch, Linux Documentation Project, March 2000.

[Dean01] Jeffrey Dean, *LPI Linux Certification In A Nutshell, a Desktop Quick Reference*, O'Reilly, June 2001, first edition.

[DigiNotar at Wikipedia.org] *Wikipedia.org: DigiNotar*, Wikipedia.

[Don99] don@sabotage.org, April 1999.

[Drake00] Joshua Drake, December 2000.

[Engelschall00] Ralph S. Engelschall, 2000.

[FreeS/WAN]

[Friedl01] Jeffrey E.F. Friedl, *Mastering Regular Expressions* , O'Reilly, 1997., ISBN 0-56592-257-3.

[Generate TLSA Record] *Huque.com: Generate TLSA Record*, Shumon Huque.

[Hinds01] David Hinds, Sourceforge.net, July 13, 2001.

[How to use systemctl] *How To Use Systemctl to Manage Systemd Services and Units*, Justin Ellingwood.

[Hubert00] Bert HubertRichard Allen, Linux Documentation Project, April 28, 2000.

[Jackson01] Ian Jackson, 24 July 2001.

[Johnson01] Richard B. Johnson, April 1999.

[Kiracofe01] Daniel Kiracofe, v1.3, January 2001.

[Krause01] Ralph Krause, January 10th, 2001.

[LPIC2sybex2nd] Christine BresnahanRichard Blum, *LPIC-2 Linux Professional Institute Certification Study Guide (Second Edition)* , Sybex, October 2016, ISBN 978-1-119-15079-4.

[LUtHL] Sander van Vugt, *Linux Under the Hood Livelessons* , Prentice Hall, Jan 13, 2017, ISBN-13 978-0-13-466299-2.

[Let's Encrypt TLSA] *Internetsociety.org: Let's encrypt certificates for mail servers and DANE*, Jan Žorž.

[Lindgreen01] Ted Lindgreen, Snow B.V., 14 september 2001.

[LinuxRef01] , Red Hat.

[LinuxRef02] , Red Hat.

[LinuxRef03] , LinuxPlanet.

[LinuxRef04] , LDP.

[LinuxRef05] , XFree86 Project Inc..

[LinuxRef06] , Apache, February 28, 2001.

[LinuxRef07] , Central Queensland University, 1999.

[LinuxRef08] , Apache.

[Liu00] Cricket Liu, Acme Byte and Wire LLC, 2000.

[Lugo00] David Lugo

[McGough01] Nancy McGough, Copyright © 1994 Nancy McGough and Infinite Ink.

[NFS] , Open Source Developer Network.

[NFSv4]

[NFSv4.2]

[NginX01] *NginX Website* , Nginx Inc., 2007.

[NginX02] *NGINX + https 101 The Basics & Getting Started* , Cloudflare, February 4th, 2016.

[Nielsen01] Mark Nielsen, Linux Gazette, February 14, 2001.

[Nielsen98] Mark Nielsen, Linux Gazette, Januari 1998.

[Nijssen99] Theo Nijssen, CERT NL, August 1999.

[Pearson00] Unknown, Squid-cache.org, August 29, 2012.

[PerlRef01] Stas Bekman, The Apache Group, September 2001.

[PerlRef02] , ActivePerl Documentation.

[PerlRef04] Randal L. SchwartzTom Phoenix, O'Reilly, July 2001.

[Poet99] Poet, Linux Review, August 21, 1999.

[PostfixGuide] Kyle D. Dent, *Postfix: The Definitive Guide* , O'Reilly, 2003, ISBN 0-000-00000-0.

[PostfixTFSreadme] *Postfix documentation : TLS Forward Secrecy in Postfix*, Postfix.org.

[PostfixTLSreadme] *Postfix documentation : Postfix TLS Support*, Postfix.org.

[Prasad01] Tanmoy PrasadChris Snell, Linux Documentation Project, July 25, 2001.

[RFC2487] *SMTP Service Extension for Secure SMTP over TLS*, Paul Hoffman.

[RFC6698] *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*, Internet Engineering Task Force.

[RFC7671] *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: Updates and guidance*, Internet Engineering Task Force.

[RIPE: BIND 10] *RIPE: The Decline and Fall of BIND 10*, Shane Kerr.

[Robbins01] Daniel Robbins, IBM DeveloperWorks, February 2001.

[Robbins01.2] Daniel Robbins, IBM DeveloperWorks, July 2001.

[Robbins96] Arnold D. Robbins, 2001.

[SMTPauthHowto] *Postfix SMTP AUTH (and TLS) HOWTO*, Patrick Ben Koetter.

[SSL01] *SSL/TLS Strong Encryption FAQ*, The Apache Software Foundation.

[SSL02] Dan Poirier, *SSL with Virtual Hosts Using SNI*, Httpd Wiki, April 2009.

[SSL03] Moxie Marlinspike, *SSL And The Future Of Authenticity*, Defcon, March 2012.

[SSL04] Moxie Marlinspike, *SSL And The Future Of Authenticity*, TLDP, January 2001.

[Sayle98] Robert P. Sayle, June 6, 1998.

[Sharpe01] Richard Sharpe, *Using Samba, Special Edition*, Que Corporation, July 2000, first printing.

[Till01] David Till

[Truemper00] Winfried Truemper, July 23, 2000.

[Tutorial on DANE and DNSSEC] *Tutorial on DANE and DNSSEC*, Wes Hardaker.

[UEFI] *Unified Extensible Firmware Interface Forum*, Various.

[USBRef01] , linux-usb.org.

[Vasudevan02] Alavoor Vasudevan, linuxdocs.org, January 2002.

[Wall01] Larry WallTom ChristiansenJon Orwant, *Programming Perl (3rd edition),* , O'Reilly, 2000., ISBN 0-596-00027-8.

[Wessels01] Duane Wessels

[Will00] Michael Will, The Linux Documentation Project.

[Wilson00] Brian Wilson, O'Reilly OnLamp.com, March 17, 2000.

[Wirzenius98] Lars WirzeniusJoanna OjaStephen Stafford, Linux Documentation Project, Version 0.7.

[Yap98] Ken Yap, Linux Focus, September 1998.

[Zadok01] Erez Zadok, *Linux NFS and Automounter Administration (Craig Hunt Linux Library),* , Sybex, 2001., ISBN 0-7821-2739-8.

[apache24upgrade] *Upgrading to 2.4 from 2.2*, The Apache Software Foundation.

[apache24upgrade] *Apache LogLevel Directive documentation*, The Apache Software Foundation.

[apache24upgrade] *What's new in Apache 2.4*, Rich Bowen.

[apachedoc] *Apache HTTP Server Documentation*, The Apache Software Foundation.

[apachesslhowto] *SSL/TLS Strong Encryption How-To*, The Apache Software Foundation.

[archDKMS] *DKMS at Arch*, Various.

[archNVMe] *NVMe*, Various.

[archUEFI] *UEFI at Arch*, Various.

[cipherli.st] *Cipherli.st: Strong Ciphers for Apache, nginx and Lighttpd* , Remy van Elst & Juerd.

[debianDKMS] *DKMS at Arch*, Various.

[digochanges] *How To Migrate your Apache Configuration from 2.2 to 2.4 Syntax*, Justin Ellingwood.

[dracut] *Dracut main project page*, Various.

[dracut] Phil Lembo, *One More Tech blog about SSSD for LDAP auth on Linux*, Wordpress.com.

[githubDKMS] *Dynamic Kernel Module System at Github*, Dell Inc..

[linuxversions] , Wikipedia.

[mozsslconf] , Mozilla.

[raymii.org] *Raymii.org Blog about SSL* , Remy van Elst.

[tomsUEFI] *Say Goodbye To Your BIOS: Hello, UEFI!*, Patrick Schmid & Achim Roos.

[tukaani] *XZ Utils on Tukaani.org*, Unknown.

[wikiCRIME] *CRIME*, Various.

[wikiDKMS] *DKMS at Wikipedia.org*, Various.

[wikiUEFI] *UEFI on Wikipedia*, Various.

[wikiXZ] *XZ on Wikipedia*, Various.

[wikipedia_apachemodules] Various, Wikipedia.

Appendix A

LPIC Level 2 Objectives

These are the LPIC Level 2 objectives, version 4.5.0, valid from February 13th, 2017.

	LPIC Level 2 Exam 201	**60**
200	Capacity Planning	
200.1	Measure and Troubleshoot Resource Usage	6
200.2	Predict Future Resource Needs	2
201	Linux Kernel	
201.1	Kernel components	2
201.2	Compiling a kernel	3
201.3	Kernel runtime management and troubleshooting	4
202	System Startup	
202.1	Customising system startup	3
202.2	System recovery	4
202.3	Alternate Bootloaders	2
203	Filesystem and Devices	
203.1	Operating the Linux filesystem	4
203.2	Maintaining a Linux filesystem	3
203.3	Creating and configuring filesystem options	2
204	Advanced Storage Device Administration	
204.1	Configuring RAID	3
204.2	Adjusting Storage Device Access	2
204.3	Logical Volume Manager	3
205	Network Configuration	
205.1	Basic networking configuration	3
205.2	Advanced Network Configuration and Troubleshooting	4
205.3	Troubleshooting Network Issues	4
206	System Maintenance	
206.1	Make and install programs from source	2
206.2	Backup operations	3
206.3	Notify users on system-related issues	1

Table A.1: LPIC Level 200 - 206 Objectives And Their Relative Weight

	LPIC Level 2 Exam 202	**60**
207	Domain Name Server	
207.1	Basic DNS server configuration	3
207.2	Create and maintain DNS zones	3
207.3	Securing a DNS server	2
208	Web Services	
208.1	Implementing a web server	4
208.2	Apache configuration for HTTPS	3
208.3	Implementing a proxy server	2
208.4	Implementing Nginx as a web server	2
209	File Sharing	
209.1	Samba Server Configuration	5
209.2	NFS Server Configuration	2
210	Network Client Management	
210.1	DHCP configuration	2
210.2	PAM authentication	3
210.3	LDAP client usage	2
210.4	Configuring an OpenLDAP server	4
211	E-Mail Services	
211.1	Using e-mail servers	4
211.2	Managing Local E-Mail Delivery	2
211.3	Managing Remote E-Mail Delivery	2
212	System Security	
212.1	Configuring a router	3
212.2	Securing FTP servers	2
212.3	Secure shell (SSH)	4
212.4	Security tasks	3
212.5	OpenVPN	2

Table A.2: LPIC Level 207 - 212 Objectives And Their Relative Weight

Chapter 14

Index

T