Web Server Management: Running Apache 2.2 under Linux

Bob Dowling
University of Cambridge Computing Service

rjd4@cam.ac.uk

Jon Warbrick
University of Cambridge Computing Service

jon.warbrick@ucs.cam.ac.uk

Web Server Management: Running Apache 2.2 under Linux
by Bob Dowling and Jon Warbrick

Installation: This course will first illustrate how to load the Apache package on a Linux server. The course uses a SUSE Linux Enterprise Server version 10 (SLES 10) system. This section of the course is Linux specific, and to a large extent SLES specific.

Configuration: The course will then demonstrate how to configure the web server *from the ground up*. The course does not teach tweaking of the default configuration but rather the writing of a configuration from scratch. The configuration will be suitable for a system running multiple virtual hosts. This part of the course applies to Apache on any platform.

These notes, and supporting material such as copies of the configuration files used in the course, are available at `http://www-uxsup.csx.cam.ac.uk/~jw35/courses/apache/` Questions from web-server administrators in the University about running Apache, or other web matters, can be sent to <`web-support@ucs.cam.ac.uk`>.

Chapter 1. Installing the software

In this chapter we will install the Apache web server package, and the packages on which it depends, and we'll briefly review the changes that doing so makes to the system. The details of this chapter are SLES-specific, but the general approach applies to any Linux system.

Installing the packages

We need to install the Apache package if we have not done so already. To do this we will have to be root. We either log in as root or **su** to root. If you **su** please be sure to use the – option to get the right environment.

```
$ /bin/su -
Password: password
#
```

SLES's Apache package is called apache2. SLES comes with various tools which can install and update packages, including **rpm**, **yast2**, and **rug**. We will use **rug**, which can fetch packages, install them and resolve package interdependencies. We assume that the system is already configured with details of a suitable installation source.

```
# rug install apache2
Resolving Dependencies...

The following packages will be installed:
  apache2 2.2.3-16.2 (SLES10-Updates)
  apache2-prefork 2.2.3-16.2 (SLES10-Updates)
    apache2-prefork-2.2.3-16.2.i586[SLES10-Updates] needed by
apache2-2.2.3-16.2.i586[SLES10-Updates]

  libapr1 1.2.2-13.2 (http://bes.csi.cam.ac.uk/install/SLES10-i386?ali...
    libapr1-1.2.2-13.2.i586[SUSE-Linux-Enterprise-Server-i386-10-0-200...

  libapr-util1 1.2.2-13.2 (http://bes.csi.cam.ac.uk/install/SLES10-i38...
    libapr-util1-1.2.2-13.2.i586[SUSE-Linux-Enterprise-Server-i386-10-...

Proceed with transaction? (y/N) y

Downloading Packages...

Transaction...

Transaction Finished
```

What happened here? First **rug** selected the most recent version of the apache2 package, and then it identified additional packages that will be needed by the one we explicitly asked it to install, so-called 'dependencies'. One of these, apache2-prefork, provides one particular flavour of the actual web server. Following confirmation from us, **rug** then downloaded and installed .

If we had access to the necessary package files, perhaps from a shared server or because we had already downloaded them, then we could have simply installed them using the **rpm**. However if we did that we'd have been responsible for selecting the most recent version of each package, and also for identifying and resolving the dependency issues.

Changes made to the system

We should quickly examine some of the changes have been made to the system by the installation of these packages. Between then, apache2 and apache2-prefork packages have installed a web server (in `/usr/sbin/`) but they have also installed files in other locations.

Locations added or changed by the installation:

- `/etc/apache2/`: this is where Apache expects to find its configuration files. Many of the files installed here are part of SLES's custom system for managing Apache, which we won't be using. However `httpd.conf` will be important because it contains Apache's main configuration. We'll be doing lots of work on this file.
- `/etc/init.d/apache2`: is the script run at system startup, and other times, to start and stop the Apache server
- `/etc/logrotate.d/apache2`: a configuration file to manage log file rotation. We'll come back to this later.
- `/usr/lib/apache2/` and `/usr/lib/apache2-prefork/`: much of Apache's functionality is implemented as a series of plug-in *modules*. Much of this course will be devoted to what they can do and how to get them to do it. Each one exists as a library file and they are stored in this directory.
- `/usr/share/apache2/`: various resources that Apache can be configured to use are stored in these directories.
- `/var/log/apache2/`: this (empty) directory is created as a place to store the log files that the web server will write as it works. An `apache2` subdirectory is used because there will typically be more than one log file in use at any time. This subdirectory holds them together. We will discuss log files in detail in Chapter 7.

In addition, the directory `/srv/www` was created by the base SLES install and provides the default location for files served by the web server. In particular, the `htdocs` subdirectory, which is initially empty, is the basic website. Anything put here will appear on the website. See the Section called *Quick and Dirty Web Server* for how to get a web site up and running as quickly as possible. Other subdirectories of `/srv/www` support software that we won't be covering in this course.

Note that the `/srv/www` directory tree is owned by root. Any changes to the website as the system currently stands need to be done by root

Programs included with the web server

- `/usr/sbin/ab2`: This a stress-tester for the web server. Please do not stress-test people's servers without their permission.
- `/usr/sbin/htdbm2`, `/usr/sbin/htdigest2`, `/usr/sbin/htpasswd2`: These manipulate user and password information for web access controls. We will see **htpasswd2** and **htdigest2** later in the Section called *Access control by client identity* in Chapter 10 but we will not be considering the other commands in this course.
- `/usr/sbin/logresolve2`: If a log file contains IP addresses rather than DNS names for clients then this program will run through the log file and write out a copy with host names replacing IP addresses. Because it caches resolved addresses it does this rather efficiently. We will be covering log files (and why they might have IP addresses rather than host names) in Chapter 7.
- `/usr/sbin/apache2ctl`: This is the script that is provided by the Apache team to simplify turning on and off the service. However, to keep the startup and shutdown of the server consistent with the rest of the system the standard startup scripts don't use this for the main work.
- `/usr/sbin/httpd2-prefork`: This is the web server itself, normally symlinked to `/usr/sbin/httpd2`.

- /usr/sbin/rotatelogs2: This is an Apache utility providing for rotating log files. However, it is not used by a SLES system because there is a system-wide log rotation facility which is used instead for consistency with the rest of the system. This will be considered in detail in the Section called *Log rotation* in Chapter 7.
- /usr/sbin/suexec2: This is a *helper program* for Apache that lets the server run external programs (e.g. CGI programs) as a different user than the user running the web service itself. As we will not be covering CGI programming in this course we will not be making any use of this program. As it is a setuid root program, you may want to remove it if you don't need it.
- /usr/sbin/apxs2: This is a tool for building Apache modules from source. We will use this in the Section called *University of Cambridge 'Raven' authentication* in Chapter 10 when we build a module to let our server use the University's *Raven* authentication service.

The '2' on the end of the names of these programs is a feature of SLES's packaging of Apache - it would allow Apache 1 and Apache 2 packages to coexist on the same server. In many other installations the commands will not have the additional '2'. They may also be in different locations, for example some may be in /usr/bin/ rather than /usr/sbin/.

Quick and Dirty Web Server

The majority of this course concerns the (re)configuration of the web server. However, we should briefly describe what a system administrator should do if he or she is happy with the default (which is not a bad set of defaults, by the way).

If the system administrator is happy to do all the changes to the web site as root then *nothing* more needs to be done other than turning on the web server.

To enable the web server (so that it gets started at system boot) the system administrator needs to use the **chkconfig** command.

```
# chkconfig --list apache2
apache2           0:off   1:off   2:off   3:off   4:off   5:off   6:off
# chkconfig apache2 on
# chkconfig --list apache2
apache2           0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

The next time the system is rebooted, the web server will be started. If you don't want to wait until a reboot, or don't want to reboot, then it can be manually started by running the script that would be run at boot time.

```
# /etc/init.d/apache2 start
Starting httpd2 (prefork)                                            done
```

If you take this easy approach then you need to know the following few facts.

- If the server's DNS name is *server* then any file placed in /srv/www/htdocs/some/path/here/file.html will be presented as URL http://*server*/some/path/here/file.html.
- Two log files will be maintained in the directory /var/log/apache2 called access_log and error_log. These will be rotated weekly or when their size reaches 4MB, and up to 99 old log files will be kept providing they are less than a year old.

You can make life much simpler for yourself (as the system administrator) if you create a group of users who are allowed to edit the document tree /srv/www/htdocs/. We cover the steps needed to achieve this later in the course.

Apache documentation

Apache comes with a large amount of documentation which many people seem to ignore! SLES provides a copy of the manual for the appropriate version of Apache in the package apache2-manual. If you install this package and use the quick and dirty approach above then a copy of the manual is available at `http://server/manual/`

```
# rug install apache2-doc
Resolving Dependencies...

The following packages will be installed:
  apache2-doc 2.2.3-16.2 (SLES10-Updates)

Downloading Packages...

Transaction...

Transaction Finished
```

If you don't have local access to a copy of the manual, current versions can always be found from the *Apache Documentation Project* (http://httpd.apache.org/docs-project/).

Chapter 2. The site's design

This chapter will describe the design of a web site that we will set as our goal for this course and discuss a small amount of theory.

Site description:

We will describe the web site that we want to create. This will be a website with a number of modern features. In particular we will demand the facility to run multiple virtual hosts (that is, different websites running off the same server).

Virtual hosting:

There will be a brief diversion while we describe exactly how virtual hosting is possible. There are a variety of different ways to achieve this goal.

The server we want

We are going to describe the server in the terms that a manager would use to describe it: rather vague. This is an excuse for this course to introduce new features one at a time.

Sites:

We are told that the server must serve two web sites. One, www.dept.cam.ac.uk, for a main department site and one, prg.dept.cam.ac.uk, for Department's 'Prevarication Research Group'; the corresponding web pages will be under the control of two different groups of users.

Facilities:

The "usual facilities" should be provided. This is too vague a specification in reality, but it is typically all the average manager will ever ask for. In this course we will assume that this means index pages, automatic directory listing, user home pages and access controls. To illustrate how to create tied-down servers we will also design the server to be this *and no more*.

Logging:

Logs should be kept for as long as possible. We will have to consider the DPA implications of this part of the specification.

One server, multiple 'virtual' hosts

There are (at least) three different ways that a single web server, and a single copy of Apache, can host more than one web site. The important issue here is how Apache can work out which site is relevant for each request.

Multiple ports:

A web server can listen on more than the default TCP port (number 80) and offer different web sites on each port. To identify a non-standard port, its number must follow the server name in the URL: http://www.dept.cam.ac.uk`:`*`port`*/some/path/here/. Apache uses the port number of the incoming query to distinguish between web sites. Beware that from some environments, particularly those behind firewalls or on wireless networks, it may be difficult or impossible to access web servers running on non-default ports.

Multiple addresses:

A single system can have more than one IP address and each can have a different web site attached to it. This leads to different server names appearing in standard URLs (i.e. there's no :*port* element in the URL) but the server names correspond to different IP addresses of the system and correspond to different web sites. Apache uses the destination IP address on the incoming query to distinguish web sites. Nowadays this approach is not encouraged because IP addresses are in short supply.

Multiple aliases

Also known as *name-based virtual hosting*, this is the most common form of virtual hosting. The server only has a single IP address, but different names in the DNS correspond to that address. So www.dept.cam.ac.uk and prg.dept.cam.ac.uk both map on to the same IP address and therefore the same server. This raises the question of how the web server can distinguish requests to the two different web sites.

Structures of HTTP queries and responses

To answer this question, we need to consider, briefly, the nature of a web request. Exactly what gets sent to a server when a URL is requested? (And for that matter, what gets sent back?)

Here's an example of what might get sent:

```
GET /index.html HTTP/1.1
Host: www.dept.cam.ac.uk
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-GB; rv:1.8.1.1)  ...
Accept: text/xml,application/xml,application/xhtml+xml,text/html;...
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

To understand name-based virtual hosting consider just the first two lines. The **GET** request only includes to the local element of the URL. The second line specifies the host name that is being asked for it.

HTTP Query Structure

`GET`

The first line declares that this is a request from a client that wishes to read information from the server. `GET` is the most common HTTP method.

`/index.html`

The second term in the first line is the *local* element of the URL requested. Note that the leading part of the URL containing the server name has been stripped out.

`HTTP/1.1`

The final element declares that the query is couched in the language of version 1.1 of the HTTP standard.

`Host: www.dept.cam.ac.uk`

The second line indicates which server the query was addressed to. It is this element of the query that allows a web server to distinguish between web sites based purely on their names, regardless of the port number(s) or IP address(es) used.

`User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-GB; rv:1.8.1.1) Gecko/20060601 Firefox/2.0.0.1 (Ubuntu-edgy)`

This optional line identifies the browser. Some servers vary the output according to this header, but you should remember that it is a *hint* and can be trivially changed on many browsers.

In this case `Mozilla` identifies the browser as one of the Netscape/Mozilla family and `5.0` ties it down to a version of Mozilla. Other information allows us to identify that it is a browser is running under Linux on an Intel platform, that it was built for the `en-GB` locale, and indicates the version numbers of the various components.

`Accept: text/xml,application/xml,application/xhtml+xml,text/html; q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5`

This specifies the formats the browser can accept and how keen it is on them. Servers can be configured to negotiate various different formats of response depending on these parameters.

text/xml,application/xml,application/xhtml+xml means that the browser is happy to accept MIME content types text/xml, application/xml, or application/xhtml+xml; otherwise it will accept text/html but the qualifier q=0.9 means that, given a choice, the browser would prefer to receive one of the earlier types (default q=1.0) than text/html. text/plain means that it can accept plain text too. The qualifier q=0.8 makes this less preferred than anything else. The browser has a general preference for image/png. Finally it will accept any format (*/*) but is not keen on them (q=0.5).

We will meet MIME content types again in Chapter 4.

`Accept-Language: en-gb,en;q=0.7,es;q=0.3`

Just as it is possible to negotiate formats it is possible to negotiate languages. A page might appear in more than one language and the browser specifies what languages it can cope with and how desirable they are. One of the authors of this document is learning Spanish and has Spanish as a third choice in the language selections after British English and any other sort of English.

`Accept-Encoding: gzip, deflate`

Just as there was negotiation over MIME content type there can also be negotiation over MIME transfer encoding. This is a mechanism for the server and browser to agree on a way to (typically) compress the data stream prior to transfer.

`Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7`

The final topic for negotiation is the character set of any text that will be sent. In this case, ISO Latin 1 is preferred, with UTF-8 and indeed everything else coming second.

`Connection: keep-alive`

This tells the server that it need not close the network connection after sending back the response to the query as other requests may be sent down the same connection. As setting up and tearing down connections are expensive operations this is a major efficiency boost.

`Keep-Alive: 300`

This instructs the server to keep the connection alive for 300 seconds in case there are any more requests. After 300 seconds of idleness the server will drop the connection.

For the record, here is the response. To make the example work, I've installed a trivial `index.html` web page. We will use this later.

```
HTTP/1.x 200 OK
Date: Wed, 21 Feb 2007 17:53:35 GMT
Server: Apache/2.2.3 (Linux/SUSE)
Last-Modified: Wed, 21 Feb 2007 17:53:33 GMT
Etag: "1c0e43-132-3caa4140"
Accept-Ranges: bytes
Content-Length: 306
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<title>The DEPT web site</title>
</head><body>
<h1>Welcome to DEPT</h1>
<p>This is the DEPT web site.</p>
</body>
</html>
</html>
```

Chapter 3. Getting started

Clean slate:

Rather than modify the existing default configuration file, we are going to delete it and start from scratch. We will start by removing the existing configuration script. This may seem dramatic but this course seeks to explain *every single line* of the configuration file that will finally be written. After we delete the file we will note that the web server won't start.

Simplest configuration:

The aim of this section is to give us enough configuration so that the server will at least start, even if it won't do anything useful.

One more line

We will add one more line to the most basic configuration file we can have to turn off many defaults so we can see them explicitly when we need them.

The least we can get away with

Deleting the configuration file is easy. Go on, you know you've always wanted to do it! What's the worst that could happen?

```
# rm /etc/apache2/httpd.conf
```

The web server will not start now. First it will complain about not having a configuration file. Perhaps we should have kept a backup...

```
# /etc/init.d/apache2 start
/apache2/httpd.conf: No such file or directory
The command line was:
/usr/sbin/httpd2-prefork -f /etc/apache2/httpd.conf              failed
```

Next, we will create an empty configuration file and see that that just changes the error message.

```
# touch /etc/apache2/httpd.conf
# /etc/init.d/apache2 start
Starting httpd2 (prefork) no listening sockets available, shutting down
Unable to open logs
startproc:  exit status of parent of /usr/sbin/httpd2-prefork: 1 failed
```

It must be admitted that as error messages go, "no listening sockets available, shutting down" is a fairly obscure way of saying "you've not told me what to do". Actually it means, "you've not told me to listen for any incoming requests so I might as well quit now".

We will start by detailing an absolutely minimal configuration file that gets the server launched but nothing else.

```
Listen  80
```

The command to tell the server to listen for connections is **Listen**. This takes one argument, specifying which interface and port to listen on. The default port assigned to web services by the Internet authorities is port 80. Quoting just a port number means to listen on that port number on *every* IP-enabled interface. Simply to launch the web server this should be all we need!

```
# /etc/init.d/apache2 start
Starting httpd2 (prefork)                                    failed
```

Unfortunately, the launched web server then immediately shuts down. By default, the web server logs error messages in an error long file. In SLES, as in many Linux distributions, this will be `/var/log/apache2/error_log`. We can look in there for clues as to what we need next.

```
[Wed Feb 21 15:54:58 2007] [alert] (2)No such file or directory:
getpwuid: couldn't determine user name from uid 4294967295,
you probably need to modify the User directive
```

What does this error message mean? It means that the web server needs to know who to run as. A standard SLES install comes pre-configured with a user wwwrun and a group www for the web server. We need to tell it to use them. This is done with the **User** and **Group** commands in the configuration file.

```
User    wwwrun
Group   www
```

While we are at it, we add one "unnecessary" line which has the effect of turning off various settings which default to being on. We do this for two reasons. The first is didactic; we want to meet these options explicitly when they become relevant rather than relying on defaults. The second is our decision to provide what was specified and no more. This line will turn off everything and we must explicitly turn on what we want.

```
Listen  80
User    wwwrun
Group   www
Options None
```

And if we start the web server now, with this four line configuration file, it launches just fine and stays running.

```
# /etc/init.d/apache2 start
Starting httpd2 (prefork)                                       done
# tail -1 /var/log/apache2/error_log
[Wed Feb 21 16:31:46 2007] [notice] Apache/2.2.3 (Linux/SUSE) configured --
resuming normal operations
# ps -ef | grep apache2
root      6377     1  0 16:31 ?       00:00:00 /usr/sbin/httpd2-prefork ...
wwwrun    6378  6377  0 16:31 ?       00:00:00 /usr/sbin/httpd2-prefork ...
wwwrun    6379  6377  0 16:31 ?       00:00:00 /usr/sbin/httpd2-prefork ...
wwwrun    6380  6377  0 16:31 ?       00:00:00 /usr/sbin/httpd2-prefork ...
wwwrun    6381  6377  0 16:31 ?       00:00:00 /usr/sbin/httpd2-prefork ...
wwwrun    6382  6377  0 16:31 ?       00:00:00 /usr/sbin/httpd2-prefork ...
root      6392  3260  0 16:34 pts/0   00:00:00 grep apache2
```

But, as the figure shows, it's not a single daemon that gets launched. There are *six* of them. The first column of the **ps** output gives the owner of the process and the second gives the process ID or *PID*. One of the server processes is owned by user root and the others by user wwwrun. That root-owned process is the parent process of all the other processes. What happens is that the startup script that we manually invoked launched the parent, root-owned process (PID 6377). It in turn launched five child processes owned by wwwrun (PIDs 6378 - 6392).

Why? Well, the idea is that the parent process does not service any request at all. Its sole purpose is to keep an eye on the child processes. If one of them dies for any reason the parent decides whether or not to replace it. (If they have all been idle for the past 48 hours it may decide that four processes are plenty.) If they are all kept too busy the parent may choose to start up some more processes to share the load. The set of child processes is called the *server pool* and is the traditional mechanism that Apache has always used to provide rapid responses. This way of working is called the *pre-forked model*.

There are other models, or MPMs (*Multi-Processing Modules*) as the Apache documentation calls them, other than pre-forked but SLES has this enabled by default and it is what we will use throughout the course.

At the moment, the server has nothing to serve. Every attempt to request a page from it results in a 404, not found error. If we look in the error log file, `/var/log/apache2/error_log` we will see the error message:

```
[Wed Feb 21 16:39:34 2007] [error] [client 131.111.10.33] File does not exist:
/srv/www/htdocs/index.html
```

which at least looks promising.

Syntax summary: Getting launched

Listen

This specifies the network interface the server should listen on. If only a port is specified then it will listen on every active network address. (Typically the system's own network address and the loopback address.)

User

This specifies the system user ID that the server process should run as. This user was created by the SLES system install.

Group

This specifies the system group ID that the server process should run as. This group was created by the SLES system install.

Options

This command sets various parameters in the configuration. We will meet these through the course as we turn them on.

Virtual hosts

However, as we are planning on hosting two web sites we ought to be thinking about two locations, one for each value of the Host: header. We should also think about what to do with requests that have neither www.dept.cam.ac.uk nor prg.dept.cam.ac.uk as the Host: header's value.

> **Use valid DNS names**
>
> The names used for the virtual hosts cannot just be made up. They must be registered in the DNS for the IP address of the server being used. Typically there will be a "real" name for the server (given by the DNS A record) and a number of aliases for the web sites (given by DNS CNAME records).

We shall create two subdirectories, `WWW` and `PRG`, of `/srv/www` for the two websites. We will also create two groups, www-admin and prg-admin, which will contain the people entitled to update the sites.

Setting up and using the `WWW` and `PRG` directories.

1. Creating the groups

```
# groupadd -r www-admin
# groupadd -r prg-admin
```

   The `-r` option on **groupadd** sets up a *system* group. These are no different from user groups in reality, but SLES assigns them from a different range of numeric IDs to keep them apart.

2. Setting up the directories

   Next we have to create `/srv/www/WWW` and `/srv/www/PRG` and set them up so that these newly created groups have sway over them. After creating the directories we need to do a number of things.

   - We must change the group of the directories. It starts out controlled by the root group.
   - We must change the permissions so that this group can add things.
   - We must set the permissions so that anything created in the directory *also* is controlled by the webadmin group.

   The change of group is done with the **chgrp** command and the two changes of permissions can be done with a single use of the **chmod** command.

```
# mkdir /srv/www/WWW
# chgrp www-admin /srv/www/WWW
# chmod g+ws /srv/www/WWW
# mkdir /srv/www/PRG
# chgrp prg-admin /srv/www/PRG
# chmod g+ws /srv/www/PRG
```

3. Adding users to the groups

We will add the users alice and bob to the www-admin group and alice only to the prg-admin group.

You can directly edit the file `/etc/group` to add users to the group line. They should be comma-separated with no spaces and no trailing comma.

```
www-admin:!:106:alice,bob
prg-admin:!:107:alice
```

Alternatively, we can use the **usermod** command to change the groups that the users are in. The `-G` option sets a user's groups.

Warning

usermod's `-G` option *sets* the user's groups. It does not add to them. You must quote *all* the user's groups. Any groups the user was previously in that are not quoted will be lost by the user.

Suppose alice is in group alpha already. Then to add her to www-admin and prg-admin we must state that she is in all the groups.

```
# usermod -G alpha,www-admin,prg-admin alice
```

If bob is in no other group, then the command used is easier.

```
# usermod -G www-admin bob
```

Note: The users will have to log in again to pick up the groups they have been added to.

To let us know we have reached the right directory we will put a file, `index.html` in each directory identifying it.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<title>The DEPT web site</title>
</head><body>
<h1>Welcome to DEPT</h1>
<p>This is the DEPT web site.</p>
</body>
</html>
```

Now we must tell the web server to use these two directories appropriately. For this we use the **NameVirtualHost** and **VirtualHost** commands.

```
NameVirtualHost *

<VirtualHost *>
ServerName      www.dept.cam.ac.uk
DocumentRoot    /srv/www/WWW
</VirtualHost>

<VirtualHost *>
ServerName      prg.dept.cam.ac.uk
DocumentRoot    /srv/www/PRG
</VirtualHost>
```

To set up a named-based virtual host we add a section like the one shown in the figure above to the configuration file. Two such sections should be added, one for www and one for prg. So what does it mean?

Syntax summary: Virtual hosts

NameVirtualHost `interface`

This instructs the web server to run name-based virtual hosts on `interface`. If the specified interface is `*` then all available interfaces are used.

<VirtualHost `interface`>

The **VirtualHost** section describes a single virtual host. Everything from the **<VirtualHost `interface`>** to **</VirtualHost>** sets parameters for a single virtual host. Parameters set outside the **VirtualHost** section normally provide default values for all virtual hosts. The interface specified must match one previously set up for named-based virtual hosting by a **NameVirtualHost** command.

ServerName

This sets the name of the server for the virtual host. If a query's Host: header matchs this then the virtual host block will be applied.

DocumentRoot

This command specifies where the server should look for its documents for the particular virtual host. This is where we get to split up our various hosts into different directories.

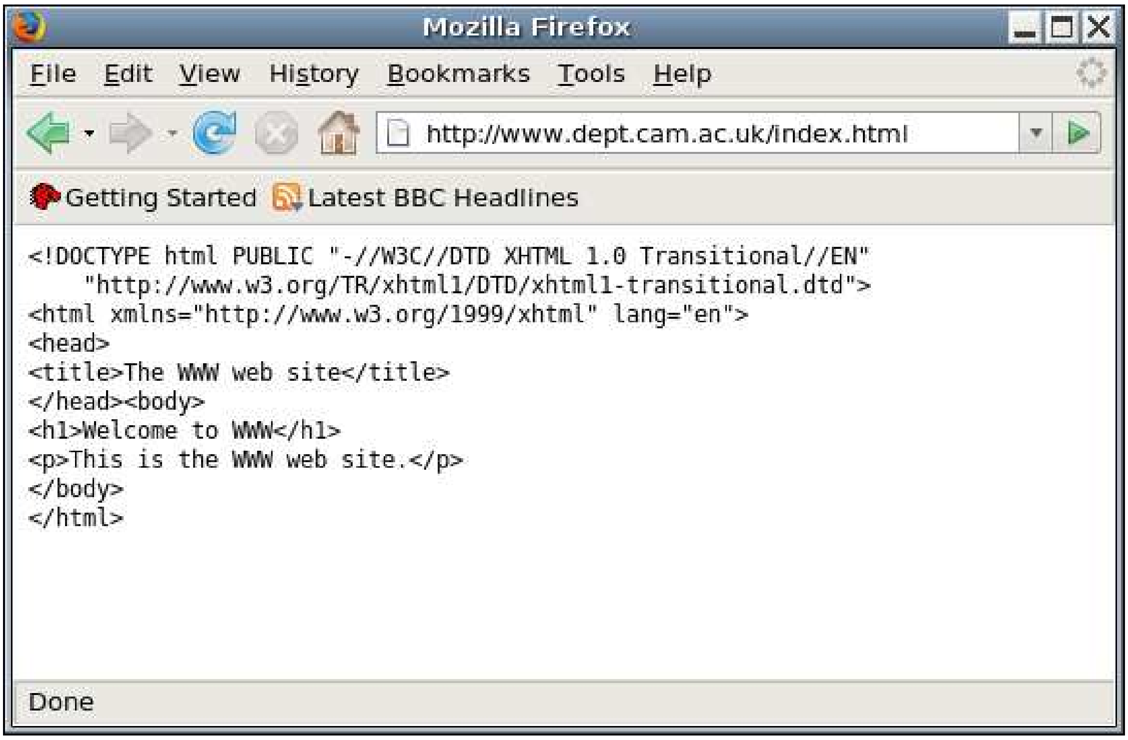
What happens if a query's Host: header doesn't match the server name of any **VirtualHost** block? Apache treats the first **VirtualHost** block in the configuration file as the default VirtualHost and will use this if nothing else matches. It can be a good idea to set Apache up with a 'dummy' first virtual host block that does nothing except display an error message saying that it's been impossible to work out which site a request was intended for.

Reloading the configuration

Apache needs to be told to re-read its configuration file before it will take account of changes, but it is not necessary to completely stop and restart our web server after each change. A rather faster mechanism is to cause it to reread its file to note changes. This can be done by using the `reload` option on the startup script.

```
# /etc/init.d/apache2 reload
Reload httpd2 (graceful restart)                            done
```

We are now running one web server supporting two web sites. However, if we request the `index.html` page from www.dept.cam.ac.uk then we get the source of the homepage and not the HTML rendering of it. We still have work to do.



Before we find out why, for completeness we should cover the assorted options that can be passed to the startup script beyond the `start`, `restart` and `reload` options we have met already.

Options to the startup script `/etc/init.d/apache2`

`start`
: Starts the web server.

`stop`
: Stops the web server.

`restart`
: Stops and starts the web server.

`try-restart`
: stop the web server and if this succeeds (i.e. if it was running before), start it again.

`status`
: Indicates whether or not the web server is running.

`full-server-status`
: Dump a full status screen; requires **lynx** or **w3m**

`reload` or `graceful`
: Causes a running web server to reread its configuration file(s) and to reopen its log files.

`help`

Not much help!

`configtest`

Does not launch a web sever but forces it to parse the configuration file for syntactic validity.

Chapter 4. Supporting MIME types

MIME types:

We will start with a very brief discussion about what *MIME types* are and in particular what *MIME content types* are. We will also see how they are associated with file suffixes in a particular system configuration file.

Modules:

We will then introduce the concept of the *module* and, in particular, the module that allows the web server to interpret MIME types.

The `Content-Type:` header

Let's take another look at the headers that get sent back by a fully configured web server.

```
HTTP/1.x 200 OK
Date: Wed, 21 Feb 2007 17:53:35 GMT
Server: Apache/2.2.3 (Linux/SUSE)
Last-Modified: Wed, 21 Feb 2007 17:53:33 GMT
Etag: "1c0e43-132-3caa4140"
Accept-Ranges: bytes
Content-Length: 306
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

In particular note the `Content-Type:` header. This identifies the document served as being of MIME content type text/html. This informs the browser that the document should be parsed as HTML rather than as plain text. This identification of content type is an important feature of HTTP that was lacking in many earlier transfer protocols.

Now let's look at the headers coming from our server as it currently stands.

```
HTTP/1.x 200 OK
Date: Wed, 21 Feb 2007 17:49:42 GMT
Server: Apache/2.2.3 (Linux/SUSE)
Last-Modified: Wed, 21 Feb 2007 17:34:30 GMT
Etag: "1c0e41-132-f8897580"
Accept-Ranges: bytes
Content-Length: 306
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

The principal difference is that the Content-Type: header now reads text/plain.

MIME types on a SLES system

So, how does the system associate MIME content types with files? There are two ways.

Content analysis

The first is to look in the file's content and deduce the MIME content type from the content.

You can see this mechanism in action with the **file** command. This command can give a "human readable" description of a file's content type or, with the `-i` option, it can give a MIME content type.

```
$ file course.pdf
course.pdf: PDF document, version 1.2
$ file course.ps
course.ps: PostScript document text conforming at level 2.0
$ file -i course.ps
course.ps: application/postscript
$ file -i course.pdf
course.pdf: application/pdf
```

The file `/usr/share/misc/magic` is used to store the information about how to map from content to MIME type. The file `/usr/share/misc/magic.mime` is used for the more verbose descriptions.

File name analysis

The other approach is to use the file name. In particular it is traditional that files should have particular suffices according to their MIME content types. This is the most commonly used approach.

This approach is taken by other utilities than just the web server and there is a system wide file giving the correspondence between file names and MIME content types. This file is `/etc/mime.types` which is part of the SLES base system (as part of the aaa-base package).

```
application/msword              doc
application/pdf                 pdf
application/postscript          ai eps ps
application/rtf                 rtf
application/x-bzip2             bz2
application/x-dvi               dvi
application/xml
audio/mpeg                      mpga mp2 mp3
image/png                       png
model/vrml                      wrl vrml
text/html                       html htm
text/plain                      asc txt
video/mpeg                      mpeg mpg mpe
video/quicktime                 qt mov
```

Apache is capable of both modes of operation. We will use the latter as it is more common. This is for historical reasons and is not a reflection on the relative values of the two mechanisms.

Loading and using the MIME module

Modules:

All the various elements of web server functionality are split out into *modules*. These are shared libraries that the web server loads when instructed to by its configuration file. In turn, the presence of a module causes new commands to be available in the configuration file corresponding to the newly available functionality.

We start by adding lines to the configuration file to load the module and identify the file containing the filename to type mapping.

```
LoadModule      mime_module     /usr/lib/apache2/mod_mime.so
TypesConfig     /etc/mime.types
```

The **LoadModule** command takes *two* arguments. The second is the filename of the shared library that it needs. It's normally possible to quote the pathname of the module relative to Apache's **ServerRoot**, often `/etc/apache2`. But SLES's Apache is built with an un-helpful **ServerRoot** of `/srv/www` so we have to use absolute pathnames here. The first argument is the name of the module within that file. Normally you need to consult the documentation to determine what a module's name is.

A list of all the common modules, together with their library file names, module names and brief descriptions is given in Appendix A at the end of these notes.

The **TypesConfig** command indicates the file that has the correspondences between file name suffixes and MIME content types.

So how does our web server work now? The pages are now presented as HTML.

Syntax summary: Loading modules

LoadModule ***`name library.so`***

Load the module named *`name`* found in the shared library file *`library.so`*.

Syntax summary: mime_module

TypesConfig ***`file`***

This command specifies the name of the file that does the lookup from filename suffix to MIME content type. On a SLES system this file is `/etc/mime.types` and is installed by the aaa-base package.

Chapter 5. Setting Options

Symbolic links

It is demonstrated that the web server does not follow symbolic links unless explicitly directed to do so.

Options FollowSymLinks

The instruction to make the server follow symbolic links is introduced.

Options

There is general discussion about the **Options** command.

Options SymLinksIfOwnerMatch

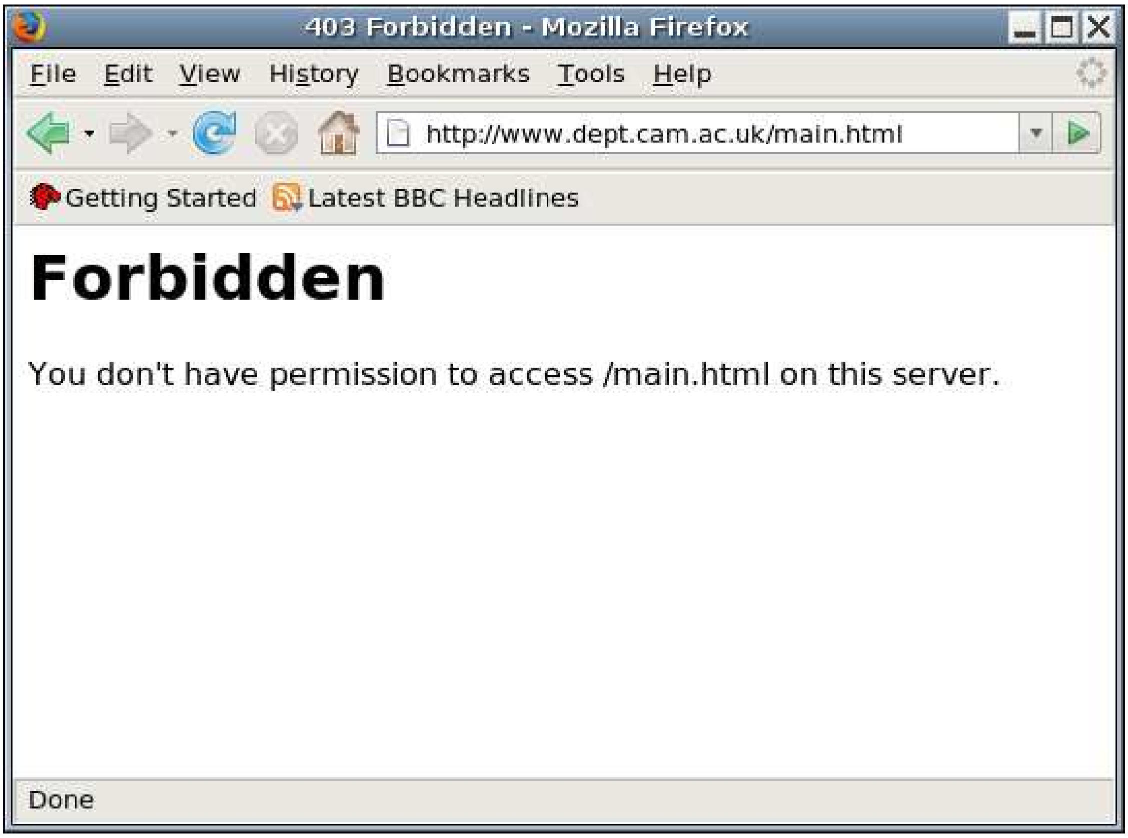
The more restrictive command is also introduced.

Symbolic links

We can now see the `index.html` file as expected but if we create a symbolic link called `main.html` to `index.html` and ask for that we get a failure.

```
$ cd /srv/www/WWW/
$ ln -s index.html main.html
$ ls -l
ls -l
total 4
-rw-r--r-- 1 root www-admin 306 2007-02-21 17:34 index.html
lrwxrwxrwx 1 root www-admin  10 2007-02-21 18:36 main.html -> index.html
```

When we try to access the symbolic link we get a 403 "Forbidden" error. The web server has found the symbolic link but has decided not to follow it.



To instruct the web server to follow symbolic links we need to set an option. You will recall we unset all options with **Options None** in the configuration file. Now we need to turn on one of them.

We can do this with the command **Options FollowSymLinks** but this has a certain subtly we need to understand. The command **Options FollowSymLinks** sets the `FollowSymLinks` option and unsets all of the others. The **Options** command followed by a list of options is *absolute*; precisely the options specified will be set and no others. For this reason we will introduce the syntax for setting (and unsetting) individual options while leaving the others unchanged.

```
Options  +FollowSymLinks
```

There is an analogous syntax with a minus sign for turning off options while leaving others untouched.

Because symbolic links might be used to circumvent access controls in the web server there is a modified version of this option with the rather unwieldy name `SymLinksIfOwnerMatch`. This instructs the web server to follow the symbolic link if and only if the symbolic link's owner (typically the user who created it) and the target's owner match.

It is worth mentioning that effects very similar to those provided by symlinks can also be created using web server facilities by using the **Alias** command which we cover later in the Section called *Improving the listings* in Chapter 6. Both have strengths and weaknesses; using both at the same time can be a recipe for madness!

Syntax Summary: The Options command

Options ***option***$_1$ ***option***$_2$ ***option***$_3$

Set options *option*$_1$, *option*$_2$ and *option*$_3$ and unset all others.

Options **+*option***$_1$ **+*option***$_2$ **+*option***$_3$

Set options *option*$_1$, *option*$_2$ and *option*$_3$ leaving all other options unchanged.

Options **−*option***$_1$ **−*option***$_2$ **−*option***$_3$

Unset options *option*$_1$, *option*$_2$ and *option*$_3$ leaving all other unchanged.

Options **+*option***$_1$ **+*option***$_2$ **−*option***$_3$

Set options *option*$_1$, *option*$_2$ and unset *option*$_3$ leaving all other unchanged.

Syntax Summary: Options for Symbolic Links

Options FollowSymLinks

The server will follow any symbolic link.

Options SymLinksIfOwnerMatch

The server will follow any symbolic link owned by the same user as its target.

Chapter 6. Handling directories

Directory URLs

Some URLs (typically those that end in /) correspond to directories rather than plain files. We need to determine how to deal with these.

dir_module

The dir_module module provides the facility of using a default file (typically called `index.html`) in a directory on behalf of the directory itself.

autoindex_module

The autoindex_module module provides the facility of generating a listing of the files and subdirectories within a directory.

Combining approaches

We will also consider how to combine the two approaches so that an `index.html` file is used if present and a directory listing is used if not.

The problem with directories

A document has a MIME content type. We've already seen that the URLs that correspond to files can now be served with the correct MIME type, so long as the file's name has the appropriate suffix. But what about directories? In particular a "top level" URL such as `http://www.dept.cam.ac.uk/` will trigger a request for /. There's no such file so the server can't serve it yet, let alone determine a MIME content type for it.

There are two solutions to this issue. The first, and simplest, is to nominate a filename (such as `index.html`) and instruct that if a directory is requested which contains a file with this name then that file should be treated as the target of the request. This is the approach we will follow in the Section called *Using default documents*.

The other approach is to give a listing of the directory's contents, typically in HTML format. We will cover this in the Section called *Automatic indexing of directories*.

Using default documents

Loading the module

We will add lines to the configuration file to load the module and to set up a default file. Because we want this functionality for both websites we place the instruction outside and before the virtual host sections.

```
LoadModule dir_module /usr/lib/apache2/mod_dir.so
DirectoryIndex index.html index.htm
```

Note that we have a new command **DirectoryIndex** which is the only additional command provided by dir_module. It is passed a list of the defaults to use if the directory is looked for. If a directory is requested then the web server will look for `index.html` in the directory because that filename is the first argument to the command. If `index.html` is missing then the server will look for `index.htm`, the second quoted name. If neither is present then the web server will give a not found error.

Remember the server must reload its configuration to pick up these new instructions.

Syntax summary: dir_module

DirectoryIndex

Specify the documents to be searched for given a directory query.

Document names are resolved relative to the directory queried unless they start with a / when they are resolved relative to the document root.

Automatic indexing of directories

The second approach we will consider is that of automatically generating a list of the entries in the directory. This used to be a popular approach to creating websites, especially those mainly consisting of files to download, but it’s now falling out of favour.

Basic listings

The relevant module is rather old and clunky, hailing back to the days when browsers didn’t support tables in HTML, but it is in very widespread use so we need to consider it. We will start by loading the module and removing dir_module (for simplicity at this stage).

This is also the largest module (in terms of number of commands) we will cover in this course. If you can cope with this one, you can cope with any of them.

```
LoadModule autoindex_module /usr/lib/apache2/mod_autoindex.so
```

If we just load the module then we see that, instead of getting a 404 “Not found” error we get a 403 “Forbidden” error instead.

This can be confusing, because 'Forbidden' is more commonly associated with access control. In this case you are seeing it because the web server has been configured to handle directories but by default won't do so. As with symbolic links above (see Chapter 5) we need to set an option to instruct the module to do its job. Note that this use of **Options** *follows* the loading of the module. Several options we will meet rely on a specific module and their use must follow the **LoadModule** line in the configuration file.

```
LoadModule autoindex_module /usr/lib/apache2/mod_autoindex.so
Options +Indexes
```

And now, if we ask for the / URL we get the list of the files and directories that appear in the top-level directory. We've included some additional files to make things more interesting.

Improving the listings

By default, the index produced is a very simple one (an itemised list in HTML). The module provides a number of commands to make it a bit more interesting. To see it in operation we will add a simple **IndexOptions** command to our configuration file to turn on "fancy indexing".

```
IndexOptions FancyIndexing
```

Next we will suppress certain rows from the listing. Why would we want to do this? Well, suppose the web developers edit their files in place (i.e. in the directory managed by the web server) with an editor (`emacs`, say) that while editing a file (`alpha.html`, say) creates work files (`#alpha.html#`) while it is running and leaves behind backup files (`alpha.html~`) when it is finished. We don't want these files appearing in the listings. We do this with the **IndexIgnore** command.

```
IndexIgnore "#*#"  "*~" ".*"
```

Note that the expressions to be ignored are placed in quotes. This is not typically necessary but under certain circumstances it is required. In this case the "#" character is the comment character in `httpd.conf` files. If it was not enclosed in quotes then everything on the **IndexIgnore** line beyond the first "#" would be ignored.

> **Warning**
>
> Just because a file name is not in the listing does not mean that it cannot be downloaded. If I see `alpha.html` and guess that there might be an `alpha.html~` I can still request it and the server will serve it to me. We will deal with blocking these downloads in the Section called *Blocking access to files* in Chapter 10.

In addition to having a listing of files, it is possible to place text above and below the listing. This can either be in the form of plain text or full-blown HTML. We will concentrate on the latter.

To add HTML above the listing the configuration must identify a *header* file. This file must have a name that identifies it as having MIME content type text/html. In the simple case, however, the file's content, should not be a full HTML document but just the HTML body component (without the leading BODY tag) for the text to appear above the listing. Everything else will be automatically generated. We identify this file (should it exist) with the **HeaderName** command.

```
HeaderName HEADER.html
```

A suitable `HEADER.html` file might look like this:

```
<p>Here is an HTML fragment.</p>

<p>It will automatically appear above the auto-generated file listing.</p>
```

Note that the `HEADER.html` file appears in the listing too. Typically this is not wanted as it is already "doing its job" by having its contents appear at the top of the page. The file `HEADER.html` would be a good candidate for the **IndexIgnore** command.

The next prettying up of the listing will be to add icons to the listing. Typically, icons are used to represent the MIME content type of the file. We will use the icons in the `/usr/share/apache2/icons/` directory which are provided for this purpose.

We are immediately presented with a problem. The `icons` directory is not in either web site's **DocumentRoot**. We could copy the directory or symlink to it, but in this case we are going to introduce another facility: aliasing. This comes courtesy of the alias_module module and its **Alias** command.

```
LoadModule alias_module /usr/lib/apache2/mod_alias.so
Alias  /icons/  /usr/share/apache2/icons/
```

The **Alias** command overrides the **DocumentRoot** for specific URLs. In this case any URL whose local part starts with `/icons/` (n.b. the trailing slash) will be looked up in `/usr/share/apache2/icons/`. If we place this directive before the definitions of the virtual hosts then it will apply to both.

Once the module has been loaded, the **Alias** command may be run multiple times, both inside and outside of the virtual host sections. If it appears within a virtual host's paragraph then it applies to just that virtual host.

The file `icon.sheet.png` in the `icons` directory gives a quick lookup of all the icons provided. Now we have access to the icons we need to know how to make use of them in directory listings. The auto-indexing module provides a slew of commands for this purpose. The trick to producing self-consistent indexes is to use as few as possible. We will set up distinct icons for the following entries.

Categories with distinct icons

- HTML web pages
- Plain text pages

- Any other “text” format
- Any image format
- Any audio format
- Any movie format
- PostScript
- Portable Document Format (PDF)
- Any other file content type
- Subdirectories
- The parent directory

The command that associates an icon with a MIME content type is **AddIconByType**. However, we will also specify the ALT text for text-based browsers with the analogous **AddAltByType** command. While we are at it, we will supply a **DefaultIcon** to use when nothing else matches.

```
AddIconByType   /icons/layout.gif       text/html
AddAltByType    "HTML file"             text/html
AddIconByType   /icons/text.gif         text/plain
AddAltByType    "Plain text"            text/plain
AddIconByType   /icons/generic.gif      text/*
AddAltByType    "Text"                  text/*
AddIconByType   /icons/image2.gif       image/*
AddAltByType    "Static image"          image/*
AddIconByType   /icons/sound1.gif       audio/*
AddAltByType    "Audio"                 audio/*
AddIconByType   /icons/movie.gif        video/*
AddAltByType    "Video"                 video/*
AddIconByType   /icons/ps.gif           application/postscript
AddAltByType    "PostScript"            application/postscript
AddIconByType   /icons/pdf.gif          application/pdf
AddAltByType    "PDF"                   application/pdf

DefaultIcon     /icons/ball.gray.gif
```

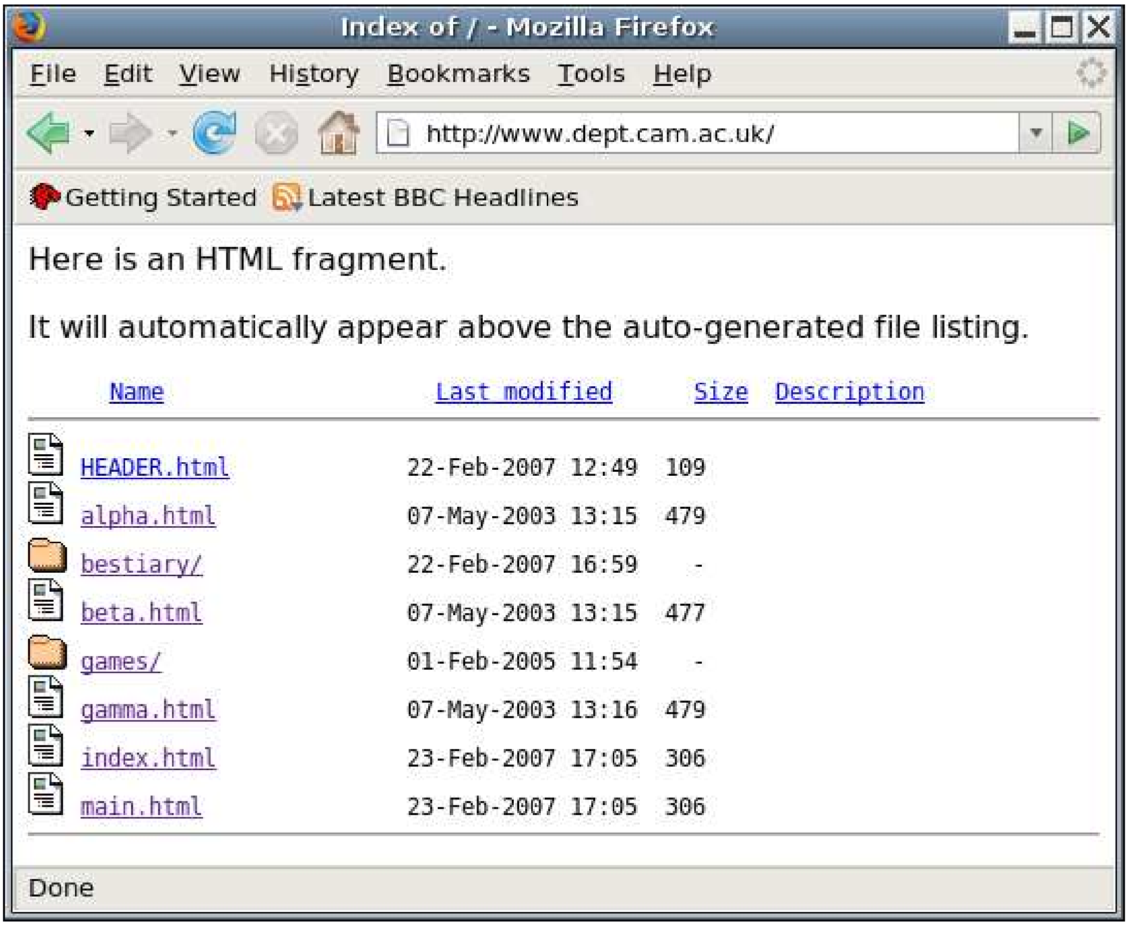
> **Note:** The icons are supplied in GIF and PNG format. Normally I would recommend using the PNG icons rather than the GIF ones since PNG is technically a better format and not troubled by patent problems. However, whoever converted the GIFs to PNGs got the background transparency wrong so you should use the GIF icons for the time being until the PNGs are fixed.

We still have a problem with directories. There is no MIME content type for a directory so we must use other facilities. The following is a filthy hack introduced by Apache version 1 and preserved into version 2.

```
AddIcon         /icons/dir.gif          "^^DIRECTORY^^"
AddAlt          "Directory"             "^^DIRECTORY^^"
AddIcon         /icons/back.gif         ".."
AddAlt          "Up"                    ".."
```

Conclusion

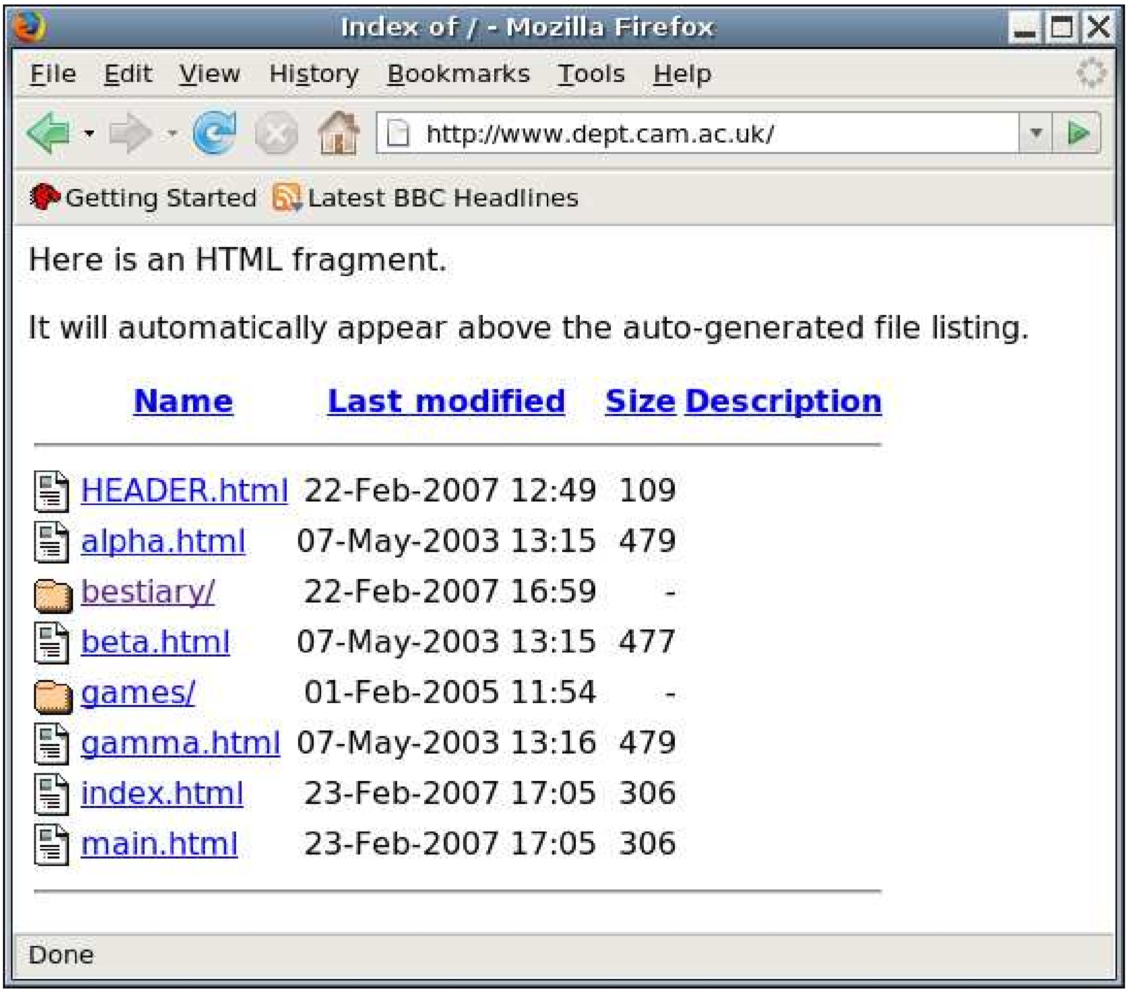
And now our listings look a bit more colourful. But this is a lot of effort for limited presentational value.



Using an HTML table

We commented above that the data presented in the file listings is inherently tabular and would be better presented as an HTML table. This is now available, as an "experimental feature" in versions of Apache beyond 2.0.23. (SLES 10 ships with 2.2.3). The authors can find no mechanisms for setting the attributes of the table from within Apache except to use stylesheets in the header file for the directory.

```
IndexOptions HTMLTable
```



Summary of the auto-indexing module

In the summary of the commands provided by autoindex_module given below only the commands and options discussed in this course are covered. There are *many* more. If you can't get the result you want with the commands given to date then consult the full Apache documentation. You might get lucky.

> **Note:** Commands and options that only make sense if fancy indexing is turned on are marked with an "(f)".

Syntax summary: autoindex_module

IndexOptions

Sets various parameters for how the index should look. The list below gives the options.

IndexIgnore "*name*" "*name*" ...

Takes a list of filenames or shell-style wildcarded filenames for file names. Files whose names match one or more of the patterns are not listed in the index.

AddIconByType *`icon`* *`mime_type`* (f)

Specifies the icon that should be used for a particular MIME content type. The MIME content type can either be fully specified (e.g. text/html) or partially specified (e.g. text/*).

AddAltByType "*`text`*" *`mime_type`* (f)

Specifies the `ALT` attribute in the `<IMG/>` tag. If you are expecting text-only browsers you might want to keep this short and of constant width (three characters is traditional). Alternatively, ditch the icons altogether.

DefaultIcon *`icon`* (f)

This specifies the icon to be used if nothing else matches. There does not appear to be an equivalent **DefaultAlt** command.

AddIcon *`icon`* *`name`* (f)

This specifies an icon for a particular file name. Typically this should be avoided but it is the best way to match the parent directory `..` and other directories with the pseudo-filename `^^DIRECTORY^^`.

AddAlt "*`text`*" *`name`* (f)

This specifies `ALT` text alongside **AddIcon**'s images.

HeaderName *`name`* *`name`* ...

This identifies the file whose contents should be placed above the file listing. The first file in the list that exists is used. These file names typically appear in the **IndexIgnore** instruction.

The files can be either plain text, an HTML body fragment or an entire "top half" of an HTML page. To stop the server adding its own HTML top half see the **IndexOptions** option `SupporessHTMLHeader`.

ReadmeName *`name`* *`name`* ...

Exactly as **HeaderName** but it corresponds to the text below the listing. This can only be plain text or an HTML body fragment.

Syntax summary: Options to IndexOptions

`FancyIndexing`

Turns on the four-column (by default) indexing mode rather than plain, bullet-list indexing mode.

`HTMLTable` (f)

This instructs Apache to use an HTML table rather than a `<PRE>` block to present the listing.

Using both modules

What happens if we run both modules? The answer is that the modules cooperate as you might hope: if there is an `index.html` file then it is used; if there isn't then the directory is indexed automatically.

```
LoadModule dir_module         /usr/lib/apache2/mod_dir.so
DirectoryIndex index.html index.htm
```

```
# Enable automatic indexing of directories
LoadModule autoindex_module   /usr/lib/apache2/mod_autoindex.so
Options +Indexes
IndexOptions FancyIndexing
...
```

Chapter 7. Logging

Error log

We will examine the error log to see what is logged and to change the amount of logging done.

log_config_module

We will load and use a module that allows us to configure exactly what we log for each request.

Log file rotation

We will illustrate the SLES system-wide mechanism for log rotation and briefly mention, and then discard, an Apache-specific way to do the same thing.

Legalities

There will be a brief description of the legal implications of keeping log files.

The error log

The error log file contains the error reports and warnings from the web server. Under SLES this defaults to the file `/var/log/apache2/error_log`. The error log is often overlooked, but when things are going wrong it will usually contain useful clues about what the problem is.

By default, the directory `/var/log/apache2` is readable only by root. You may want to change this on your system. Within the directory, the files are created world-readable. Only the directory's permissions need be changed.

Let's consider a number of the typical entries in the error log as it currently stands.

```
[Thu Feb 22 13:14:55 2007] [notice]
Graceful restart requested, doing restart
[Thu Feb 22 13:14:55 2007] [notice] Apache/2.2.3 (Linux/SUSE) configured --
resuming normal operations
```

Our first example will be seen in the log files from this course more than any other lines (we hope!). The line that starts "`Graceful restart requested`" is the logged entry that means we requested a reload of the configuration file.

The line that (hopefully) follows it is the line from Apache that says it has been (re)configured and that it is "resuming normal operations", i.e. serving web pages again.

```
[Thu Feb 22 15:13:35 2007] [notice] caught SIGTERM, shutting down
```

SIGTERM is an operating system signal with a universal function. It is the instruction to a process to shut down.

```
[Thu Feb 22 15:14:43 2007] [error] [client 131.111.10.33] File
does not exist: /srv/www/WWW/nonsuch.html
```

Another commonly occurring `error_log` line is the one corresponding to a request for a file that does not exist.

So far, the `error_log` lines we have looked at have had a standard format:

- [*date*]
- [*severity*]
- *message*

We can change the level of the logging (of formatted messages) with the **LogLevel** command. Either globally, or within specific virtual hosts' sections we can issue the command **LogLevel debug**, say, to get more debugging.

Level	Meaning	Example
`emerg`	System is unusable. The entire web service has failed.	"`Child cannot open lock file. Exiting`"
`alert`	Action must be taken immediately. The entire web service has either failed already or is imminent danger of failing completely.	"`getpwuid: couldn't determine user name from uid`"
`crit`	Critical condition. A child server has failed and others are likely to follow.	"`socket: Failed to get a socket, exiting child`"
`error`	A request on a single request.	"`File does not exist: /var/www/WWW/nonesuch.html`"
`warn`	Warnings.	"`child process 10108 still did not exit, sending a SIGTERM`"
`notice`	Normal behaviour but worthy of notice.	"`Apache/2.0.40 (Red Hat Linux) configured -- resuming normal operations`"
`info`	Purely informational	"`Server seems busy, (you may need to increase StartServers, or Min/MaxSpareServers)...`"
`debug`	Debugging.	"`Opening config file ...`"

Messages issued from a running web server are well formatted. However, if you make a syntax error in the `httpd.conf` file then the server won't launch and the error message is rather more stark.

```
Syntax error on line 21 of /etc/apache2/httpd.conf:
Invalid directory indexing option
```

It is also possible to move the error log file, or to do without the file altogether (but still log errors).

```
LogLevel info
ErrorLog /var/log/apache2/error.log
```

The **ErrorLog** directive gives the name of the error log file. If the file name given is "`syslog`" then logging is not done to `/var/log/apache2/syslog` but rather all error logs are passed to the operating system's system logger. This can be useful if you arrange for your system logs to be transmitted off-machine to a central logging engine which you want to receive Apache error logs too.

Finally, if the file name starts with a pipe character, |, then what follows is interpreted as a command which should be forked and executed and which has the error log lines passed to it on its standard input.

Syntax summary: Error logging commands

ErrorLog ***`logfile`***

If *`logfile`* begins with "`/`" then this file will be used for the error log.

If *`logfile`* is "`syslog`" then the error logs will be passed to the system logger with facility local7.

If *`logfile`* is "`syslog:`*`facility`*" then the error logs will be passed to the system logger with facility *`facility`*. This must be one of the facilities known to the system logger; you can't just make up your own.

If *`logfile`* begins with "|" then what follows will be run as a command to receive the log lines on standard input.

Anything else is interpreted as a filename relative to the server root.

LogLevel ***`level`***

Any errors generated at logging levels equal to or more serious than *`level`* will be logged.

Access logs

To date the only log file we have met is the error log. There is no logging in our current server configuration when things aren't going wrong. Probably we want to log the requests that are made of our server. These are the access logs.

We need to decide what we want to log and where to log it to. We may want more than one log file for different sets of data.

As (almost) ever, the means to get this functionality is to load a module: log_config_module from `mod_log_config.so`.

This provides us with one particularly useful command: **CustomLog**. This allows us to specify what information to record and where to record it for each query/response at the server. This power comes at the price of almost complete syntactic obscurity at first glance. But in all honesty it's not that bad.

Suppose we wanted to record just the following information about each query processed by the server:

- Time & date
- URL requested
- Name of system making the request
- Whether the request was successful

```
LoadModule      log_config_module       modules/mod_log_config.so
CustomLog       logs/request.log        "%t %U %h %s"
```

Each of the elements beginning with a percentage character is called an *escape code* and is converted into some piece of logged information. A complete list of the codes is given in Appendix B.

Four escape sequences

%t

The time of the request

%U

The URL requested

%h

The client hostname

%s

Status code of the request

To illustrate what they indicate and what they don't, we will request three URLs and note a number of problems in the logged output.

The requested URLs

- `http://www.dept.cam.ac.uk/`
- `http://prg.dept.cam.ac.uk/`
- `http://www.dept.cam.ac.uk/gamma.html`

```
[22/Feb/2007:15:51:43 +0000] /index.html 131.111.10.33 304
[22/Feb/2007:15:52:04 +0000] /index.html 131.111.10.33 304
[22/Feb/2007:15:52:19 +0000] /gamma.html 131.111.10.33 304
```

Notice how requests for the top-level directory are being logged as requests for `/index.html`. It appears that apache applys at least some of it's internal processing to the value it loggs with %U, in this case using dir_module to locate a suitable index document.

Problems with the output as it stands

- There is no record of whether www.dept.cam.ac.uk or prg.dept.cam.ac.uk was queried.
- The hostnames are addresses, not names

The simplest way to address the issue of which website was queried is to move the **CustomLog** lines into the virtual host sections and to have two different log files. This gives them the flexibility to log different things too.

```
<VirtualHost *>
ServerName www.dept.cam.ac.uk
DocumentRoot /srv/www/WWW
CustomLog /var/log/apache2/www.log "%t %U %h %s"
</VirtualHost>
```

If we really wanted a single log file with the virtual host information we could use the %v escape code to record it.

To enable the use of hostnames rather than addresses, we must arrange for the web server to do DNS lookups for the IP addresses on each incoming query. We will do this with the **HostnameLookups** command. This command is a core Apache command and not part of the logging module. In some circumstances, hostnames will be looked up even without this command. For example if you do any access controls based on host names as we will be in the Section called *Access control based on client IP address* in Chapter 10. We will set this on globally. If either website wanted to record IP addresses rather than hostnames then it can do so by using %s rather than %h.

```
HostnameLookups On
```

This results in log entries that look like this:

```
[22/Feb/2007:15:59:06 +0000] /index.html mnementh.csi.cam.ac.uk 200
[22/Feb/2007:15:58:40 +0000] /alpha.html mnementh.csi.cam.ac.uk 200
[22/Feb/2007:15:58:47 +0000] /beta.html mnementh.csi.cam.ac.uk 200
```

Common Log Format

A standard format for access log files is used by many utilities and replied on by the web analysis programs. It is called "The Common Log Format" (CLF) and is shown below.

```
CustomLog /var/log/apache2/www.log "%h %l %u %t \"%r\" %>s %b"
```

This produces log entries that look like this:

```
mnementh.csi.cam.ac.uk - - [22/Feb/2007:16:01:48 +0000] "GET / HTTP/1.1"
200 306
mnementh.csi.cam.ac.uk - - [22/Feb/2007:16:01:53 +0000] "GET /nonsuch.html
HTTP/1.1" 404 210
```

The escape sequences used in the Common Log Format

%h

The client's hostname

%l

If the web server was doing IDENT (*RFC1413* (ftp://ftp.rfc-editor.org/in-notes/rfc1413.txt)) lookups then the returned userid would be here.

%u

If the client had authenticated as a particular user for this request the userid would be recorded here. We discuss authentication in detail in the Section called *Access control by client identity* in Chapter 10.

%t

The time of the request.

%r

The first line of the query.

%>s

The status code finally returned to the client. There is a subtle difference between %s and %>s. In most cases they are identical. In cases where the URL gets remapped then %s gives the status code of the initial lookup and %>s the code of the final lookup (and the code passed back to the client).

%b

The number of data bytes (i.e. excluding the headers) sent back to the client in the case of successful completion.

Named formats

A common requirement is for all virtual hosts to log in the same format. To assist with this it is possible to name a format definition and to then refer to the format's name in the **CustomLog** line.

```
LogFormat     "%h %l %u %t \"%r\" %>s %b"  clf

<VirtualHost *>
ServerName www.dept.cam.ac.uk
DocumentRoot /srv/www/WWW
CustomLog /var/log/apache2/www.log clf
</VirtualHost>
```

Logging headers

One very useful escape code is %{*fubar*}i which will log the value of incoming header *fubar*. We could use this as %{Host}i to record the queried Host header, for example, to check our virtual hosting was working as expected.

The values of incoming headers are used to define a common alternative to "Common Log Format". This is called "Combined Log Format" since it combines the Access Log with the Referer and User Agent logs that were maintained in separate files by early versions of Apache.

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

Additional escape sequences used in the Combined Log Format

%{Referer}i

The value of the 'Referer:' header which most browsers include in requests. This contains the URL of the page that contain the link the user queried to bring them here. This can be particularly useful to find remote pages that have bad links onto your site.

%{User-Agent}i

The value of the 'User-Agent:' header which most browsers include in requests. If present, this normally identifies the browser being used, though it can be trivially changed on most browsers.

Log rotation

It is one thing to create logs; it is quite another to cope with them. A log file grows without bound unless action is taken and this can cause problems.

Problems with growing log files

- Larger files are harder to manipulate.
- File systems run out of space.
- The information you log may constitute personal data.

A solution to this generic problem of log file growth is *log rotation*. This involves the regular (nightly or weekly, typically) moving of an existing log file to some other file name and starting fresh with an empty log file. After a period the old log files get thrown away.

Because this is a general issue, many Linux distributions (SLES included) include a general solution that can be applied to any set of log files, not just the web server's.

There is an Apache-specific solution (which is provided by the **rotatelogs** command) but we will use SLES's generic solution, provided in the logrotate package.

Once each night the **logrotate** program reads in its configuration files telling it which logs to rotate and how to do it. One of these files tells it to rotate Apache's log files.

The main configuration file sets up the defaults and then reads in a directory of instructions for specific sets of log files from the `/etc/logrotate.d` directory.

```
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# uncomment this if you want your log files compressed
#compress

# uncomment these to switch compression to bzip2
compresscmd /usr/bin/bzip2
uncompresscmd /usr/bin/bunzip2

# former versions had to have the compresscommand set accordingly
#compressext .bz2

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp -- we'll rotate them here
#/var/log/wtmp {
#    monthly
#    create 0664 root utmp
#    rotate 1
#}

# system-specific logs may be also be configured here.
```

`/etc/logrotate.conf`: commands

weekly

Each file should be rotated weekly. The log rotation job runs nightly, though, so this can be changed to daily for a specific log file if desired.

The three commands that specify how often rotation should take place are **daily**, **weekly** and **monthly**.

rotate 4

Keep four sets of log files. The comment is slightly inaccurate; four *weeks'* worth of logs will be kept if rotation is done weekly. If rotation is done daily then this command means that four *days'* worth of logs are kept.

create

After moving the main log file from `logfile` to `logfile.1` a new, empty `logfile` should be created.

include /etc/logrotate.d

This command instructs the log rotation program to read in every file in this directory. One of these files will correspond to the web server's log files.

The `/etc/logrotate.d/apache2` file (part of the apache2 package, not the logrotate package), contains the instructions specific to the web server logs.

```
/var/log/apache2/access_log {
    compress
    dateext
    maxage 365
    rotate 99
    size=+4096k
    notifempty
    missingok
    create 644 root root
    postrotate
     /etc/init.d/apache2 reload
    endscript
}

/var/log/apache2/error_log {
    compress
    dateext
    maxage 365
    rotate 99
    size=+1024k
    notifempty
    missingok
    create 644 root root
    postrotate
     /etc/init.d/apache2 reload
    endscript
}
```

`/etc/logrotate.d/apache2`: commands

/var/log/httpd/access_log { ... }

This specifies that the commands within the curly brackets are to be applied to the access log. A similar block applies to the error log.

compress

Rotated log files should be compressed. Logfiles, which can be very large, typically compress very well.

dateext

Archive old versions of log files adding a date extension like YYYYMMDD to the filename.

maxage 365

Remove rotated logs older than 365 days. The age is only checked if a logfile is to be rotated.

rotate 99

Delete old log files once they have been rotated 99 times.

size=+4096k

Rotate the file if it becomes bigger than 4MB, even if it was last rotated less than a week ago.

notifempty

This command instructs the system not to rotate the logs if the current main log file is empty. See the discussion below about whether this is a good idea or not.

missingok

This is the instruction not to return an error if a particular log file is not present.

create 644 root root

After rotation, create a new logfile owned by user root and group root and with read/write permissions for it's owner and read permissions for everyone else.

postrotate ... endscript

Following the rotation of a log file the commands between **postrotate** and **endscript** are run. This rotation program runs as root so care must be taken with the commands that appear here.

/etc/init.d/apache2 reload

This instructs Apache to do a "Graceful Restart" in which the master web server demon advises its children to exit after their current request and then re-reads its configuration files and re-opens its log files.

There are two points which must be made about log rotation and changing its settings. These are to do with the presence of editor backup files and the Data Protection Act (1998).

Backup files

The command **include /etc/logrotate.d** will read in *almost every* file in that directory. So if you edit the file `apache2` and leave behind both `apache2` and `apache2.old` then *both* these files will be included and your log files will have the log rotation process applied twice. Now, because of the **weekly** (or **monthly** or **daily**) commands the rotation shouldn't actually happen but it is still not certain that the right file will be applied.

Data Protection Act (1998)

The contents of the log files may constitute personal data. If, along with any information stored within the University, they identify individuals then they definitely do. As a result, you must have a privacy policy, *and stick to it*. For this reason, the author recommends very strongly that you remove the **notifempty** command. If your policy says that personal data is only kept for 28 days and it is kept for a week longer because one week's log files happened to be empty then you are in breach of your privacy policy.

Chapter 8. Users' own web pages

userdir_module

We will introduce the relevant module and the single command it provides.

Simple use

We will start with the simplest provision of users' personal web pages by loading the module and using it in its simplest form.

Complex use

We will then give an example of how it can be used to redirect lookups to an entirely different system.

The principle of this chapter is to provide your users with the ability to create their own web pages. The web pages may be located on the servers in question, or on a different server altogether. As ever, there is a module that provides the extra functionality. In the example below, we provides user pages in all the virtual hosts.

```
LoadModule      userdir_module /usr/lib/apache2/mod_userdir.so
UserDir         public_html
```

In this example, the command **UserDir public_html** causes any URL of the form `http://`*`server`*`/~bob/path/name.html` to correspond to a file `~bob/public_html/path/name.html`. (In this filename the expression "`~bob`" means "the home directory of user bob" and is standard Unix notation. It is this that the URL notation is based on.

The directory name `public_html` is not fixed and can be modified. Furthermore, more complex mappings of username onto file name can be provided. Any instance of "`*`" in the "directory name" will be replaced with the user's ID.

If the directory name is actually a URL then, instead of the web server looking for a local directory, it responds with an HTTP redirection, pointing the web client at a new URL, typically on a different server.

UserDir argument	Translated path
`public_html`	`~bob/public_html/alpha/beta.html`
`www`	`~bob/www/alpha/beta.html`
`/var/www/users`	`/var/www/users/bob/alpha/beta.html`
`/var/www/*/web`	`/var/www/bob/web/alpha/beta.html`
`http://`*`elsewhere`*`/users`	`http://`*`elsewhere`*`/users/bob/alpha/beta.html`
`http://`*`elsewhere`*`/*/web`	`http://`*`elsewhere`*`/bob/web/alpha/beta.html`
`http://`*`elsewhere`*`/~*/`	`http://`*`elsewhere`*`/~bob/alpha/beta.html`

It is possible to give a sequence of targets to the **UserDir** command. In this case they will be searched in turn until one provides the server with the file or directory it is looking for. Only the last entry in the list is allowed to be a redirection to another server (i.e. a URL) because when the server reaches this one it sends back the redirection to the browser and never gets to discover if the file existed at the far end.

Note that the Apache user, wwwrun under SLES, must be able to read files if it is to serve them. If it can't read the files in a user's `public_html` directory then all this isn't going to work.

Chapter 9. Delegated control

<Directory>

Applying a specialised set of commands just to a subdirectory of a web site from the `httpd.conf` file.

Include

Splitting the main configuration file into components.

AccessFileName

Nominating a filename to handle control from the directory itself.

AddDescription

A commonly used command in the delegated configuration files to set the Description column in automatic indexes.

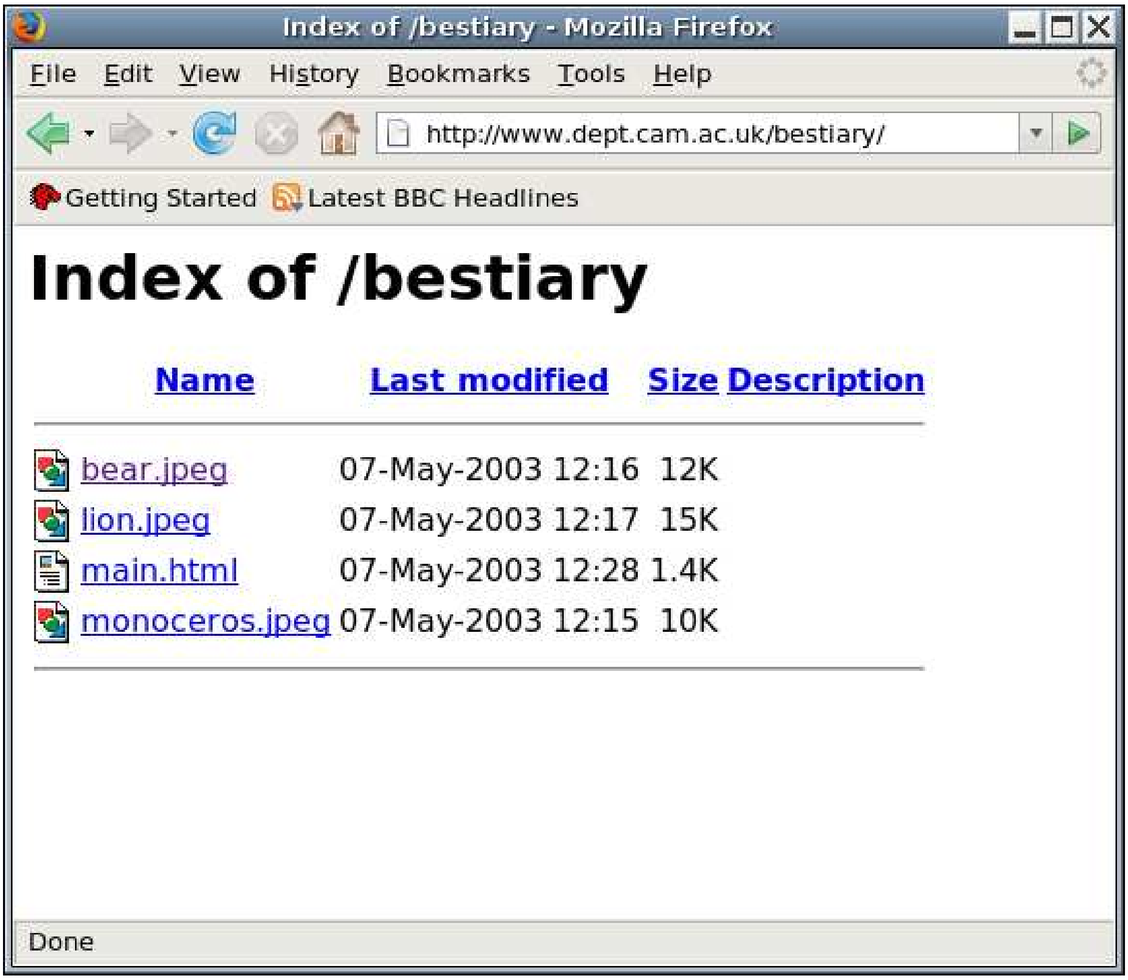
AllowOverride

Controlling what can be delegated

Customising only part of a web site

The user directory example was the first where we were passing control outside our tidy server root. It may well be that we want a different configuration for these, relatively uncontrolled, areas.

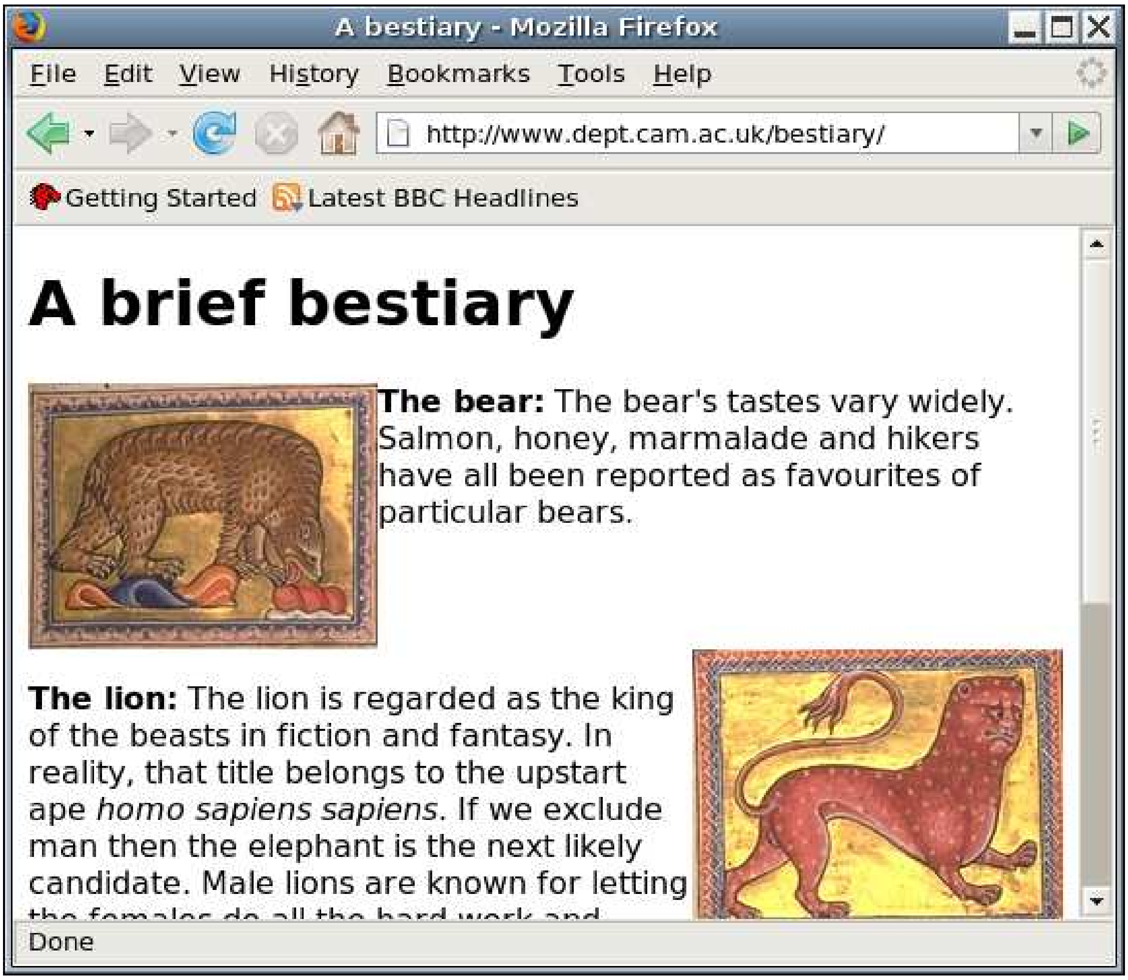
There are a number of aspects to this. You must decide what defaults you want passed to these areas and what you want absolutely fixed. You also need to know how to override the defaults where permitted.



We will start by noting how to change settings from within the `httpd.conf` file for a directory tree. In our current configuration file the directory index file name is `index.html`. Suppose for a subdirectory of one of our web sites we wanted to change it to be `main.html`. How would we do that?

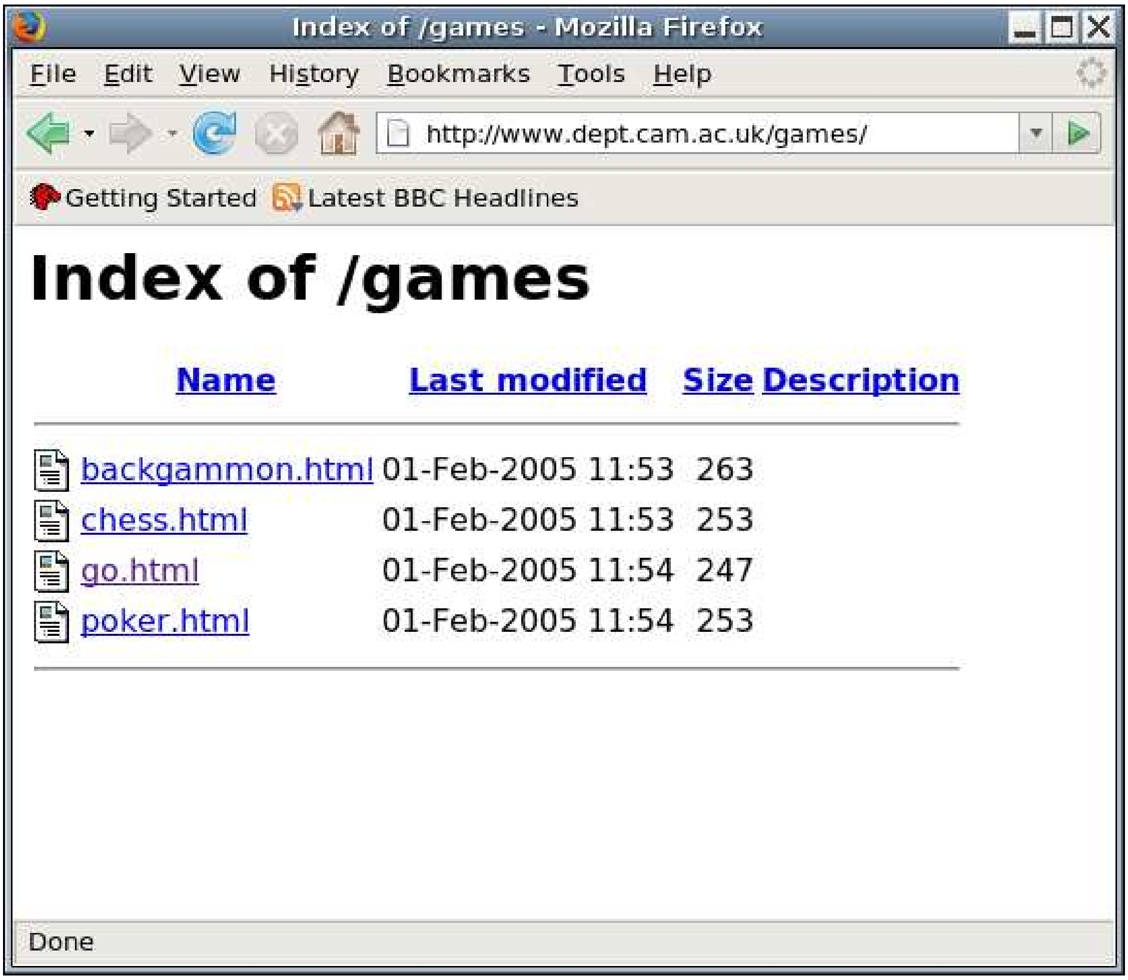
```
<VirtualHost *>
ServerName www.dept.cam.ac.uk
DocumentRoot /srv/www/WWW
CustomLog /var/log/apache2/www.log clf

<Directory /srv/www/WWW/bestiary>
  DirectoryIndex main.html
</Directory>

</VirtualHost>
```

The **<Directory *dir*>** ... **</Directory>** identifies a series of commands which should override or enhance the general settings for a specific subdirectory, `/var/www/WWW/bestiary` in the example given in the figure.



In the case of commands we have met, it is easy to imagine simply issuing them again within a **<Directory>** block to override the previous settings. But what about turning features on or off? A common example is to turn on or off the automatic generation of indexes.

At the moment we can see the index of the `games` directory in the www.dept.cam.ac.uk web site.



We can turn indexing off with the **Options** command as follows.

```
<VirtualHost *>
ServerName www.dept.cam.ac.uk
DocumentRoot /srv/www/WWW
CustomLog /var/log/apache2/www.log clf

<Directory /srv/www/WWW/bestiary>
  DirectoryIndex main.html
</Directory>

<Directory /srv/www/WWW/games>
  Options -Indexes
</Directory>

</VirtualHost>
```

And any future attempt to index `games` gives a 403, Forbidden, error.

Using Access Files

But this isn't delegating control. We have allowed for variation in subdirectories but we have not truly delegated the controls to anyone who cannot rewrite the configuration file and tell the server to reread it. We need a means to delegate control of a subdirectory into the subdirectory itself.

The **AccessFileName** command names a file (or set of files) that will be looked for within the directory being served and whose contents will be regarded as if they had been inside a **<Directory>** block for that specific directory. The name of the command tells of its origins; it was used to set the access rights for a directory tree. It is, however, a fully generic delegated configuration, not just delegated access control.

The default file name used, `.htaccess`, also reflects its history as a delegated access control mechanism. It is also traditionally a "dot file" to hide it from the index listings. It's far better to list the file name in a **IndexIgnore** statement and to give it a plain file name so the conventional Unix utilities will actually show you it's there.

So we could copy the contents of the **<Directory /var/www/WWW/games>** block to `/var/www/WWW/games/configuration` and the contents of the **<Directory /var/www/WWW/bestiary>** block to `/var/www/WWW/bestiary/configuration`.

There may be certain properties that you don't want the users messing with. To this end there is limited support for restricting what the users can override with their delegated configurations. This is controlled via the **AllowOverride** command. For the time being we will allow users to mess with everything.

```
AccessFileName  configuration
IndexIgnore     configuration
<Directory /srv/www/>
  AllowOverride All
</Directory>
```

A suitable `configuration` for the `/games/` directory might be:

```
DirectoryIndex      main.html index.html
```

This puts the control of the files in the hands of the people who have access to the directories.

Once configured, these files are searched for and read by Apache every time it accesses the directories in which they appear. This has the advantage that you don't have to restart the web server to make changes to them visible (which is vital since the people that use these files don't normally have the rights to restart Apache). The downside is that Apache has to do more work to serve every request.

The **AllowOverride** command is rather unsatisfactory - it allows the controllers of `httpd.conf` to stop the **Options** command being used in the **AccessFileName** files, but not to specify which options can and can't be set. It can specify what you can do with **IndexOptions** but not whether or not you can enable/disable indexes at all. It has many limitations.

Simple uses of AllowOverride

AllowOverride None

The delegated configuration files aren't even read. Their content is entirely ignored.

AllowOverride All

Any command that is syntactically legal in the delegated configuration file is allowed to have effect.

AllowOverride Options

The delegated configuration file is allowed to run the **Options** command. There is no mechanism to control which of its arguments are permitted.

AllowOverride Indexes

The delegated configuration file is allowed to run the **IndexOptions** command and all the commands that modify the index. This does not permit the use of **Options [+|-]Indexes**; you need **AllowOverride Options** for that.

Chapter 10. Access control

Two ways

There are two ways to do access control: by the location of the client and by the identity of the user operating the client.

Client location

There is a brief discussion of why this mechanism is fraught with difficulties caused by proxies and the like. Then the commands to implement it are covered.

User identity

There is a discussion of the Basic and Digest protocols for user identification. Access by user or group and user administration is then covered.

Raven

The University's Raven Web Authentication System is briefly described.

Mixed working

The mixed case of authorising passwordless access from within the institution but requiring authentication from outside will be given in detail.

Blocking names and directories

Application of access control to block access to files with particular names, and to entire directories, is discussed.

Two posibilities

Now we move to the topic of access control. There are fundamentally two ways of doing this: by client location and client identity.

Client location involves specifying whether access is permitted based on the IP address or hostname of the client (i.e. browsing) system. When a request is received by the server the IP address from which the request was received is known. This address, or the hostname associated with it in the DNS, is checked against a set of rules to determine whether or not the request should be honoured.

Proxy servers

Client location security is often used within the University for restricting access to an institution or to the University, loosely defined as "anything in cam.ac.uk". This approach doesn't work but is often regarded as "good enough" to keep happy the politicians, lawyers and other people who don't understand technology. From the point of view of the web administrator it also has the advantage of simplicity. The reason it doesn't work is that *web proxies* can forward a request from outside Cambridge on to a server within Cambridge which sees the request coming from within Cambridge and honours it. The Computing Service has had its internal minutes cached on Google for the whole world to read after a web proxy on the CS staff network went unnoticed.

External users

It's increasingly common for members of the University to use computers not connected to the University network - people with broadband at home, people working from wireless networks while travelling, etc. In some cases people use such connections almost exclusively and this trend can be expected to increase. Security based on client location denies these users access to information which they are intended to be able to see.

Client identity involves challenging the user to quote some means of identifying him or herself before permitting access to the document requested. This has the advantage of dealing with proxies, but the disadvantage of requiring administration of the userids and passwords. A common compromise is to create a *single* userid and password for a set of pages and pass the pair on to anyone who needs access. This has the disadvantage that you don't know which of your users read the pages, but often you don't want to know.

To avoid the password administration problem, the Computing Service provides a central authentication system that web administrators can use if the want. This allows members of the University to identify themselves using a centrally administered user-id and password.

Access control based on client IP address

As ever, this functionality is provided by a module: authz_host_module from library `mod_authz_host.so` (in versions of Apache before 2.2 this was access_module from library `mod_access.so`).

This module offers us three commands: **Allow**, **Deny** and **Order**.

These restrictions need not cover the whole server, and typically don't. They are in any case examples of commands which must be placed within a **<Directory>** block or delegated configuration file. If you want to cover the whole web server then the document root must be used for the **<Directory>** block.

The **Order** command takes one of two arguments: `Deny,Allow` and `Allow,Deny`. No whitespace is allowed around the comma. While it may look like a comma-delimited list it is not; it is just a pair of rather strange looking arguments that have a comma as one of their characters.

If the argument is `Deny,Allow` (the default) then the initial state is that all access is allowed, then *all* the **Deny** statements are processed and then they are overridden by the **Allow** statements.

If the argument is `Allow,Deny` then then the initial state is that all access is prohibited, then all the **Allow** statements are processed and then they are overridden by the `Deny` statements. This is best illustrated with some examples.

```
LoadModule authz_host_module  /usr/lib/apache2/mod_authz_host.so
<Directory /srv/www/WWW/bestiary>
  Order Allow,Deny
  Deny   from touble.csi.cam.ac.uk
  Allow  from cam.ac.uk
  #Deny  from csi.cam.ac.uk
</Directory>
```

Here is what happens when a request from `trouble.csi.cam.ac.uk` is processed.

Stage	Match?	State
Initial		All requests refused.
Allow from cam.ac.uk	Rule matches.	Access is allowed.
Deny from trouble.cam.ac.uk	Rule matches.	Access is denied.
Final		Access is denied.

Note that the sequence of **Allow** and **Deny** commands is unimportant and that their processing is entirely dependent on the **Order**.

The addresses given in the **Allow** and **Deny** statements can be specified in a variety of ways. The examples given are for the **Allow** command but are equally applicable to the **Deny** command.

Syntax summary: Options on the Allow command

Allow from cam.ac.uk

Access is allowed from any host whose name ends with cam.ac.uk.

Allow from 131.111.11.148

Access is allowed for queries originating from 131.111.11.148. Note that any query redirected through a web proxy or cache will have the address of the web proxy or cache.

Allow from 131.111

Access is allowed for queries originating from IP addresses whose first two bytes are 131.111. Note that Cambridge has more networks than just this primary one.

Allow from 131.111.10.0/255.255.254

Access is allowed from any IP address which when masked by 255.255.254.0 gives 131.111.10.0.

Allow from 131.111.10.0/23

Access is allowed from any IP address whose first 23 bits form the address 131.111.10.0.

Access control by client identity

The alternative mechanism for restricting access to web pages is to demand a userid and password from the user.

HTTP "Basic Authentication"

1. Browser sends request for a web page.
2. Server sends back a 401 error code and specifies a realm.
3. Browser prompts user for userid and password for the realm.
4. User quotes userid and password.
5. Browser *repeats the initial request* with an extra header quoting the userid and password.
6. Server sends the page if the userid/password are OK.
7. Browser sends request for another web page.
8. Server sends back a 401 error code and specifies the same realm.

9. Browser recovers the userid and password it has for that realm and repeats the initial request with the extra header.
10. Server sends the page.

Of course things are different if the userid and password don't grant access to the page. There are two ways this can happen. The user and password could match but that user, now identified, might not be allowed access to the page. In this case the server sends back a 403, Forbidden, error code. Alternatively, the userid and password might not match, in which case the server sends back the 401 code again and the cycle of prompting the user repeats.

What we need to know is how to set up the server so that userids and passwords are known to the server and certain pages are flagged as requiring user authentication.

To start with, we will need some modules: auth_basic_module, authn_file_module and authz_user_module (in versions of Apache before 2.2 this functionality was all provided by auth_module). We will then specify a mechanism to identify users and finally specify policies regarding which identified users are allowed access.

So, first we need to identify users. This comes in two parts: the first involves setting up userids and passwords at the server end and the second involves telling the web server to use these for identifying users.

The userids and passwords are *not the same as the login IDs*. Indeed, they will often not be login IDs at all. They are maintained with a distinct file which we will need tools to manipulate. This file is traditionally called `htpasswd` though we have flexibility regarding its name and location. A server administrator must also decide whether to have a single password file for the whole server or one per virtual host (or even for each subtree of the virtual host he wants to restrict access to). Granting a user a userid and password noes *not* automatically assign that userid rights to access pages (though we can configure policy so that it does). In this example, we will work with a single userid/password file for both virtual hosts. It's a shortcoming of the Unix permissions model that we cannot specify that a file should be writable by members of either one group or another. We will use a webadmin group to control access to this file. Note that the file should not be servable by the web server.

```
# groupadd -r webadmin
# usermod -G www-admin,prg-admin,webadmin bob
# mkdir /etc/apache2/access
# chgrp webadmin /etc/apache2/access
# chmod g+ws /etc/apache2/access
# ls -ld /etc/apache2/access
drwxrwsr-x 2 root webadmin 4096 2007-02-23 12:16 /etc/apache2/access
# touch /etc/apache2/access/passwd
# chmod g+w /etc/apache2/access/passwd
# ls -l /etc/apache2/access/passwd
-rw-rw-r-- 1 root webadmin 0 2007-02-23 12:17 /etc/apache2/access/passwd
```

We make the *directory* writable rather than just the individual files to make life easier for programs that move files about within directories for backing up.

```
$ htpasswd2 -m /etc/apache2/access/passwd bob
New password: password
Re-type new password: password
Adding password for bob
$ cat /etc/apache2/access/passwd
bob:$apr1$kEDyP/..$n0DCjezTD.T.C.1s3td6..
```

htpasswd's `-m` option causes the password file to use an *MD5* password encoding for the password. This is better than the traditional (and default) *crypt* algorithm. This makes the password much harder to reverse engineer from the file but all userid/password schemes are vulnerable to dictionary attacks and it is important that the password file not be downloaded to make this attack much harder.

Now that we have a way to identify users we need to specify policies. As with authz_host_module the restrictions on access can only be specified in a **<Directory>** block or in a delegated configuration file.

The simplest policy, called "valid user" is to permit access to any user who can authenticate against the web password file.

```
LoadModule      auth_basic_module /usr/lib/apache2/mod_auth_basic.so
LoadModule      authn_file_module /usr/lib/apache2/mod_authn_file.so
LoadModule      authz_user_module /usr/lib/apache2/mod_authz_user.so
LoadModule      authz_user_module /usr/lib/apache2/mod_authz_user.so

<Directory /srv/www/WWW/bestiary>
  AuthType      Basic
  AuthName      "Restricted area"
  AuthUserFile  /etc/apache2/access/passwd
  Require       valid-user
</Directory>
```

Syntax summary: implementing the "valid user" policy

<Directory /var/www/>...</Directory>

This is the standard block for restricting a set of commands to a directory tree.

The commands in this block could appear in a delegated configuration file.

AuthType Basic

This defines the protocol used for the exchange of userid and password. Every browser supports this protocol, but it does send passwords in plain text. A superior protocol, called "Digest" exists and is supported by modern browsers. See the Section called *Variations on a theme of user identification* for details.

AuthName "Restricted area"

This identifies the realm applying to the files in the directory tree. This string appears in the challenge for the userid and the password and is used by the browser to work out which previously given userid and password it should send without having to prompt the user again.

AuthUserFile /etc/apache2/access/passwd

This identifies the file used to contain userids and passwords. *This cannot be the system `/etc/passwd` file!*

Require valid-user

This specifies the policy. Any user validated against the password file may access the pages.

Given this setup (and a reload of the server's configuration file) we can see the effect it has on our web server. Our attempt to access the `index.html` page results in a challenge for userid and password.

Note that the prompt contains the phrase “Restricted area”. That text comes directly from the **AuthName** command. If we fill in any valid userid and password from the `/etc/apache2/access/passwd` file we can proceed.

Next we will consider other policies. We will assume that we have created three additional web userids: tom, dick and harry.

```
<Directory /srv/www/WWW/bestiary>
  AuthType       Basic
  AuthName       "Restricted area"
  AuthUserFile   /etc/apache2/access/passwd
  Require        user bob tom
</Directory>
```

The **Require user bob tom** statement replaces the “valid user” policy with a “one of these users” policy.

If you plan to use certain collections of users repeatedly for access control this scheme can be taken further and groups of users can be defined. We can then specify that the validated user be one of a series of groups.

First we must define our groups. We will create a groups file this time by hand because there are no tools analogous to **htpasswd** to manage the files for us.

```
managers:   bob tom
workers:    dick harry
```

We also need a module that knows about group files: authz_groupfile_module (in versions of Apache before 2.2 this functionality was part of auth_module). We can then change from a user list to a group list by specifying which group file to use and which groups are permitted access.

```
LoadModule      authz_groupfile_module /usr/lib/apache2/mod_authz_groupfile.so

<Directory /srv/www/WWW/bestiary>
  AuthType       Basic
  AuthName       "Restricted area"
  AuthUserFile   /etc/apache2/access/passwd
  AuthGroupFile  /etc/apache2/access/group
  Require        group managers
</Directory>
```

Syntax summary: Require

Require

The **Require** command specifies the *policy* of who is allowed access once identification is complete.

Require valid-user

Any authenticated user may have access to the pages.

Require user ***user*** $_1$ ***user*** $_2$ ***user*** $_3$ **...**

Only one of the listed users may have access to the pages.

Require group ***group*** $_1$ ***group*** $_2$ ***group*** $_3$ **...**

Any user in one or more of the listed groups may have access to the pages.

If we wanted to delegate policy regarding access control by this mechanism we must allow the override with **AllowOverride AuthConfig**.

Variations on a theme of user identification

What we described in the previous section is *a way* to provide user authenticated access control. We used the Basic protocol and simple text files to store the userids, passwords and groups.

The Basic protocol can be replaced with the Digest protocol. This comes from module auth_digest_module from `mod_auth_digest.so`.

```
LoadModule      auth_digest_module     /usr/lib/apache2/mod_auth_digest.so
LoadModule      authn_file_module      /usr/lib/apache2/mod_authn_file.so
LoadModule      authz_user_module      /usr/lib/apache2/mod_authz_user.so
LoadModule      authz_groupfile_module /usr/lib/apache2/mod_authz_groupfile.so

<Directory /srv/www/WWW/bestiary>
  AuthType      Digest
  AuthName      "Restricted area"
  AuthDigestDomain /
  AuthUserFile  /etc/apache2/access/digest_passwd
  AuthGroupFile /etc/apache2/access/group
  Require       group managers
</Directory>
```

The password file is replaced with one with a different structure, but the group file is the same as it was before.

```
$ touch /etc/apache2/access/digest_password
$ htdigest /etc/apache2/access/digest_password "Restricted area" bob
Adding user bob in realm Restricted area
New password: password
Re-type new password: password
```

The other issue we mentioned was that text files were used to hold the users, passwords and groups. For a small number of users this is fine but if your users reach into the thousands you may want to consider alternatives that are faster to search. Alternatively, you may already have an LDAP authentication mechanism and want to use that. A series of other modules exist for providing authentication with passwords and groups held in other formats.

University of Cambridge 'Raven' authentication

Allocating new user names and passwords for access to sites is a problem. It requires users to remember additional passwords, and requires administrators to create, issue, re-issue and revoke accounts. Worse, basic authentication is insecure in the face of an attacker who can monitor networks. Raven, a central authentication system provided by the Computing Service, attempts to address these issues.

Under Apache, Raven is implemented by ucam_webauth_module, but since this doesn't come with Apache we first need to collect a copy and build it. Building the module requires the apache2-devel and openssl-devel packages which we'll also have to install if they are not already available. We also need a copy of the public key that ucam_webauth_module uses to validate responses from the central Raven server.

Building and installing ucam_webauth_module

1. Install the apache2-devel and openssl-develpackages

```
# rug install apache2-devel openssl-devel
Resolving Dependencies...

The following packages will be installed:
  apache2-devel 2.2.3-16.2 (SLES10-Updates)
  libapr1-devel 1.2.2-13.2 (SLES10-Base)
    libapr1-devel-1.2.2-13.2.i586[SLES10-Base] needed by apache2-devel...

  libapr-util1-devel 1.2.2-13.2 (SLES10-Base)
    libapr-util1-devel-1.2.2-13.2.i586[SLES10-Base] needed by apache2-...

  openssl-devel 0.9.8a-18.13 (SLES10-Updates)

Proceed with transaction? (y/N) y

Downloading Packages...

Transaction...

Transaction Finished
```

2. Building and installing the module

   The Raven module, and other Raven resources, is available from the *Raven project pages* (http://raven.cam.ac.uk/project/).

```
# tar zxf mod_ucam_webauth-1.4.0.tar.gz
# cd mod_ucam_webauth-1.4.0
# apxs2 -c -i -lcrypto mod_ucam_webauth.c
/usr/lib/apr-1/build/libtool --silent --mode=compile gcc -prefer-pic -
O2 -march=i586 -mtune=i686 -fmessage-length=0 -Wall -D_FORTIFY_SOURCE=
2 -g -fPIC -Wall -fno-strict-aliasing -DLDAP_DEPRECATED  -DLINUX=2 -D_
REENTRANT -D_GNU_SOURCE -D_LARGEFILE64_SOURCE -DAP_DEBUG -pthread -I/u
sr/include/apache2  -I/usr/include   -I/usr/include/apr-1   -c -o mod_
ucam_webauth.lo mod_ucam_webauth.c && touch mod_ucam_webauth.slo
/usr/lib/apr-1/build/libtool --silent --mode=link gcc -o mod_ucam_webaut
h.la  -lcrypto -rpath /usr/lib/apache2 -module -avoid-version    mod_uca
m_webauth.lo
/usr/share/apache2/build/instdso.sh SH_LIBTOOL='/usr/lib/apr-1/build/lib
tool' mod_ucam_webauth.la /usr/lib/apache2
/usr/lib/apr-1/build/libtool --mode=install cp mod_ucam_webauth.la /usr/
lib/apache2/
cp .libs/mod_ucam_webauth.so /usr/lib/apache2/mod_ucam_webauth.so
cp .libs/mod_ucam_webauth.lai /usr/lib/apache2/mod_ucam_webauth.la
cp .libs/mod_ucam_webauth.a /usr/lib/apache2/mod_ucam_webauth.a
ranlib /usr/lib/apache2/mod_ucam_webauth.a
chmod 644 /usr/lib/apache2/mod_ucam_webauth.a
PATH="$PATH:/sbin" ldconfig -n /usr/lib/apache2
----------------------------------------------------------------------
Libraries have been installed in:
   /usr/lib/apache2

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the `-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the `LD_LIBRARY_PATH' environment variable
```

```
      during execution
    - add LIBDIR to the 'LD_RUN_PATH' environment variable
      during linking
    - use the '-Wl,--rpath -Wl,LIBDIR' linker flag
    - have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------
chmod 755 /usr/lib/apache2/mod_ucam_webauth.so
```

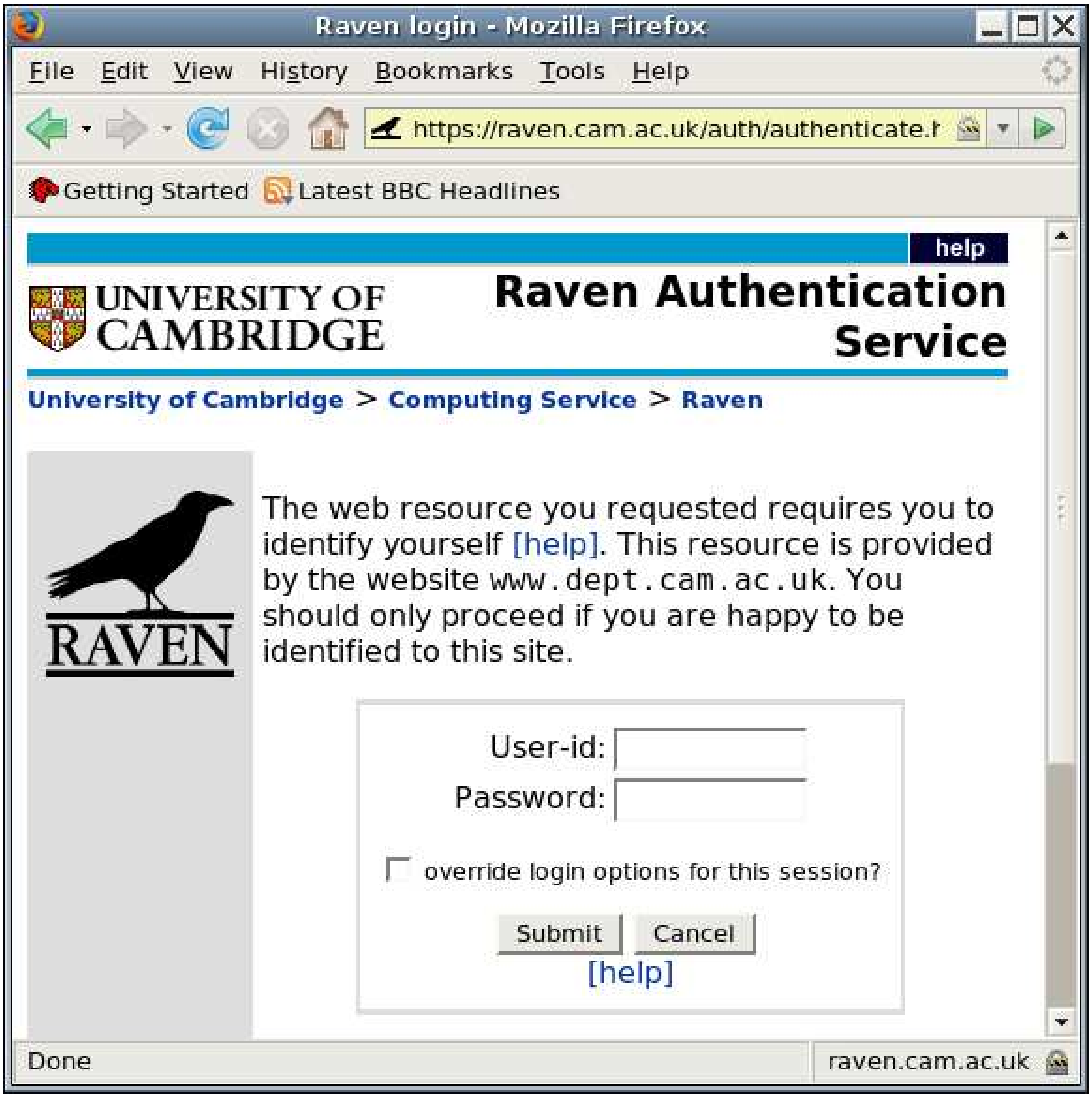
3. Installing the Raven key

   Raven uses Public Key Cryptography to prevent replies from the central servers from being forged. To validate these replies, ucam_webauth_module needs access to the current Raven public key.

   ```
   # mkdir /etc/apache2/webauth_keys
   # cp pubkey2 /etc/apache2/webauth_keys
   ```

ucam_webauth_module is configured much like other authentication modules. It relied on the services of the standard authz_user_module for to control user access and on authz_groupfile_module for group file support so you must load them as well. ucam_webauth_module also needs a random string to validate cookies that it sets so you must provide that as well.

```
LoadModule      authz_user_module      /usr/lib/apache2/mod_authz_user.so
LoadModule      authz_groupfile_module /usr/lib/apache2/mod_authz_groupfile.so

LoadModule      ucam_webauth_module    /usr/lib/apache2/mod_ucam_webauth.so
AACookieKey     "now is the time for all good chickens to be counted"
AAKeyDir        /etc/apache2/webauth_keys

<Directory /srv/www/WWW/bestiary>
  AuthType      Ucam-WebAuth
  AuthGroupFile /etc/apache2/access/group
  Require       group managers
</Directory>
```



Syntax summary: implementing Raven

AACookieKey "*some string*"

A random key used to protect cookies from tampering. Any unpredictable string is fine. This key must be kept secret, since with knowledge of the key an attacker can forge authentication.

AAKeyDir *path*

Pathname of a directory containing the public keys used by ucam_webauth_module. In many cases this defaults to something sensible under `/etc/apache2`; this isn't the case with SLES's Apache build.

AuthType Ucam-WebAuth

Selects Raven authentication.

Mix and match: Location and Authentication

This brief section shows how the two mechanisms for controlling access, location and identification, inter-operate. Specifically, there is a common desire in the University to grant passwordless access from within the department or cam.ac.uk domain and passworded access otherwise.

Suppose we wanted our website to be accessible from cam.ac.uk without a password and with Raven authentication from elsewhere.

```
<Directory /srv/www/WWW/bestiary>
  Order         Allow,Deny
  Allow         from cam.ac.uk
  AuthType      Ucam-WebAuth
  AuthGroupFile /etc/apache2/access/group
  Require       group managers
  Satisfy       any
</Directory>
```

The two worlds of access control are joined by the **Satisfy** command. This has two possible options: `Any` and `All`. **Satisfy Any** requires the request to satisfy either the location requirement or the authentication requirement. **Satisfy All** would require it to satisfy both.

Blocking access to files

There is one last aspect of access control we must consider. We have stopped certain files being listed in indexes in the Section called *Automatic indexing of directories* in Chapter 6 but we warned that this did not stop the files being downloaded if the client could guess the name. This section will demonstrate how to block downloads of files matching certain expressions in the same way as the **IndexIgnore** command stops files matching those patterns being listed.

We can restrict certain commands to files that match regular expressions with the **`<FilesMatch>` ... `</FilesMatch>`** directive. We can put a simple denial of all access in this block.

In an ideal world, **IndexIgnore** and **`<FilesMatch>`** would accept the same syntax for describing their files. Unfortunately they don't, and this is a serious flaw in the Apache Software Foundation's way of handling their modules. **IndexIgnore** uses shell-style wildcards, formally known as *globbing*, and **`<FilesMatch>`** uses **sed**-style *regular expressions*.

Our current example configuration file has the line

```
IndexIgnore  "#*#"  "*~"  "configuration"
```

and the equivalent **`<FilesMatch>`** regular expression is

```
(^#.*#$|.*~$|^configuration$)
```

An apropriate configuration would be:

```
<FilesMatch (^#.*#$|.*~$|^\..*|^configuration$)>
  Order  allow,deny
  Deny   from All
</FilesMatch>
```

It's also possible to block access to whole directories and directory trees. For example we don't want anyone to access any information outside `/var/www/`, `/usr/share/apache2/icons/` and `/home/`*`user`*`/public_html`. While the current configuration only allows access to these directories, it's possible that a mistake in the future could mess this up. A better approach is to deny access to everything by default and then to explicitly all access as required. While we are at it, we'll also turn off **Options** and **AllowOverride** by default and only enable them as needed.

```
<Directory />
  Order Allow,Deny
  Deny from all
  Options None
  AllowOverride None
</Directory>

<Directory /srv/www>
  Order allow,deny
  Allow from all
  Options FollowSymlinks Indexes
  AllowOverride All
</Directory>

<Directory /home/*/public_html>
  Order Allow,Deny
  Allow from all
  Options Indexes
</Directory>

<Directory /usr/share/apache2/icons/>
  Order Allow,Deny
  Allow from all
  Options Indexes
</Directory>
```

Chapter 11. Conclusion

Tidying up

Some re-ordering of the final configuration file.

What's next?

A taster of other Computing Service courses, and of additional features available in standard Apache.

A finished configuration file

We have illustrated a number of facilities in the Apache 2.2 web server which can be used to create a web server serving multiple web sites.

The configuration file we have built as we go along is syntactically valid, but reflects its didactic origins. Our final act will be to tidy it up.

The first thing typically done is to move all the **LoadModule** commands to a block near the start of the file. That allows us to use all their commands in whatever order we want further down the file.

The next thing we would do is to reorder the commands to exploit this freedom. In our case the only major shuffle will be to put the **IndexIgnore** statement next to the **FilesMatch** block that cover the same files. We have also added some magic to make the Apache manual available under `/manual/`, moved some of the "deny by default" rules to the top of the file, and indented the content of the various blocks for readability.

```
Listen 80
User wwwrun
Group www

# Load the modules needed for this file
LoadModule            mime_module  /usr/lib/apache2/mod_mime.so
LoadModule             dir_module  /usr/lib/apache2/mod_dir.so
LoadModule       autoindex_module  /usr/lib/apache2/mod_autoindex.so
LoadModule           alias_module  /usr/lib/apache2/mod_alias.so
LoadModule      log_config_module  /usr/lib/apache2/mod_log_config.so
LoadModule         userdir_module  /usr/lib/apache2/mod_userdir.so
LoadModule      authz_host_module  /usr/lib/apache2/mod_authz_host.so
LoadModule      authz_user_module  /usr/lib/apache2/mod_authz_user.so
LoadModule      auth_basic_module  /usr/lib/apache2/mod_auth_basic.so
LoadModule      authn_file_module  /usr/lib/apache2/mod_authn_file.so
LoadModule authz_groupfile_module  /usr/lib/apache2/mod_authz_groupfile.so
LoadModule    ucam_webauth_module  /usr/lib/apache2/mod_ucam_webauth.so
LoadModule        setenvif_module  /usr/lib/apache2/mod_setenvif.so
LoadModule     negotiation_module  /usr/lib/apache2/mod_negotiation.so

# Deny-by-default
<Directory />
  Order Allow,Deny
  Deny from all
  Options None
  AllowOverride None
</Directory>

<Directory /srv/www>
  Order allow,deny
  Allow from all
```

```
  Options FollowSymlinks Indexes
  AllowOverride All
</Directory>

# Set up MIME content type recognition
TypesConfig /etc/mime.types

# Set up default documents for directory queries
DirectoryIndex index.html index.htm

# Enable automatic indexing of directories
Options +Indexes

IndexOptions FancyIndexing
HeaderName HEADER.html

# Suppress indexing and access to backup and working files
IndexIgnore "#*#"  "*~" ".*" "configuration"
<FilesMatch (^#.*#$|.*~$|^\..*|^configuration$)>
  Order  allow,deny
  Deny   from All
</FilesMatch>

# Set up access to standard icons
Alias  /icons/  /usr/share/apache2/icons/
<Directory /usr/share/apache2/icons/>
  Order Allow,Deny
  Allow from all
  Options Indexes
</Directory>

# Set up icons and associated alt text
AddIconByType   /icons/layout.gif       text/html
AddAltByType    "HTML file"             text/html
AddIconByType   /icons/text.gif         text/plain
AddAltByType    "Plain text"            text/plain
AddIconByType   /icons/generic.gif      text/*
AddAltByType    "Text"                  text/*
AddIconByType   /icons/image2.gif       image/*
AddAltByType    "Static image"          image/*
AddIconByType   /icons/sound1.gif       audio/*
AddAltByType    "Audio"                 audio/*
AddIconByType   /icons/movie.gif        video/*
AddAltByType    "Video"                 video/*
AddIconByType   /icons/ps.gif           application/postscript
AddAltByType    "PostScript"            application/postscript
AddIconByType   /icons/pdf.gif          application/pdf
AddAltByType    "PDF"                   application/pdf

DefaultIcon     /icons/ball.gray.gif

AddIcon         /icons/dir.gif         "^^DIRECTORY^^"
AddAlt          "Directory"         "^^DIRECTORY^^"
AddIcon         /icons/back.gif         ".."
AddAlt          "Up"          ".."

# Display file listings as a table
IndexOptions HTMLTable

# Set the error logging level to "info": informational messages.
# The default is "warn": warnings.
LogLevel info

# Specify the error log file - error.log in place of error_log
ErrorLog /var/log/apache2/error.log
LogFormat       "%h %l %u %t \"%r\" %>s %b"   clf
```

```
# We want hosts' names rather than address in the logs
HostnameLookups On

# Users' own web pages
<Directory /home/*/public_html>
  Order Allow,Deny
  Allow from all
  Options Indexes
</Directory>
UserDir         public_html

# Delegate control via "configuration" files
AccessFileName  configuration
IndexIgnore     configuration
<Directory /srv/www/>
  AllowOverride All
</Directory>

# Raven authentication config
AACookieKey     "now is the time for all good chickens to be counted"
AAKeyDir        /etc/apache2/webauth_keys

<Directory /srv/www/WWW/bestiary>
  Order         Allow,Deny
  Allow         from cam.ac.uk
  AuthType      Ucam-WebAuth
  AuthGroupFile /etc/apache2/access/group
  Require       group managers
  Satisfy       any
</Directory>

# Access to the Apache manual
AliasMatch ^/manual(?:/(?:de|en|es|fr|ja|ko|ru))?(/.*)?$ \
                                          "/usr/share/apache2/manual$1"
<Directory "/usr/share/apache2/manual">
    Options Indexes
    AllowOverride None
    Order allow,deny
    Allow from all
    <Files *.html>
        SetHandler type-map
    </Files>
    SetEnvIf Request_URI ^/manual/(de|en|es|fr|ja|ko|ru)/ prefer-language=$1
    RedirectMatch 301 ^/manual(?:/(de|en|es|fr|ja|ko|ru)){2,}(/.*)?$ \
                                                             /manual/$1$2
</Directory>

# Set up name-based virtual hosting on all interfaces.
NameVirtualHost *

<VirtualHost *>
ServerName www.dept.cam.ac.uk
DocumentRoot /srv/www/WWW
CustomLog /var/log/apache2/www.log clf
</VirtualHost>

<VirtualHost *>
ServerName prg.dept.cam.ac.uk
DocumentRoot /srv/www/PRG
CustomLog /var/log/apache2/prg.log clf
</VirtualHost>
```

What's next?

Having completed this course you are now in a position to follow up by adding modules that provide for extra facilities. The computing service has two follow-on courses from this one that build on this foundation.

Follow-on web server courses

Web Server Management: Securing Access to Web Servers

This course introduces the use of the HTTPS (secure http) protocol used to protect communication between web browsers and web servers. This additional security is particularly appropriate when sensitive information is being transmitted (passwords, credit card numbers, personal details, etc) or when the identity of an end-user needs to be established securely.

The course is presented from the point of view of a web server administrator who wishes to configure servers to support such communication. It provides an overview of the https protocol and of related cryptographic components, including public key and symmetric key encryption, message digests and digital certificates as they relate to HTTPS. It also describes how to obtain certificates for web servers, and demonstrates the configuration of an Apache server to use HTTPS.

CGI Scripting for Programmers: Introduction

The *Common Gateway Interface* (CGI) underlies much of the modern web. It provides the most popular way by which web servers can respond to browser requests by invoking programs and using the resulting output as their response. CGI programs can make decisions based on information contained in browser requests, and so can be used for various tasks, including dynamic page generation, processing fill-in forms, interfacing to databases, implementing 'shopping carts', etc.

This two-afternoon course covers the CGI itself, various aspects of HTML and HTTP that are directly relevant to CGI programmers, and general CGI programming issues including security. CGI programs can be written in a variety of languages, and their use is supported by most web servers. This course uses the Perl scripting language to develop examples for use with an Apache web server running under Linux. However the vast majority of the material covered is applicable to other languages and servers, and should be readily understandable by anyone with programming experience. Some of the examples may be suitable for general use.

In addition, there are a whole range of other features that are available in Apache "out of the box" that may be of use in some circumstances.

Additional Apache features

Performance tuning

There are a huge range of configuration commands for customising Apache's performance. While the default settings of all these are fine for a small personal site they will probably need adjusting for anything bigger.

Content negotiation

Apache can take advantage of facilities in HTTP to allow it to automatically serve different content to different users. This is handled by an automatic process of negotiation and can include varying languages, character encodings and content types.

Customisable Error pages

All the error messages produced by Apache can be customised.

Rewrites

Apache can automatically rewrite URLs. This can be particularly useful for ensuring that old URLs continue to work following site reorganisation.

Header manipulation

Various facilities exist to control the HTTP headers that are sent with documents, including control of document expiry, automatically adding cookies, etc.

Server-side includes

Documents can be automatically modified as they are served to include other files, dynamic information such as the date, etc.

Proxying and Caching

Apache can be configured as an HTTP proxy, optionally also providing caching facilities.

Server information

Additional modules allow the Apache server itself to be monitored in real time.

Appendix A. Apache modules

This lists the modules shipped with SLES's packages.

Table A-1. Modules shipped as part of the base apache2 package.

Library	Module name	Description
`mod_actions.so`	actions_module	Run specific CGI programs according to the MIME content type of the object served.
`mod_alias.so`	alias_module	Override the **DocumentRoot** directive for specific URLs.
`mod_asis.so`	asis_module	Allows the HTTP headers to be in the file, rather than generated automatically by the web server.
`mod_auth_basic.so`	auth_basic_module	User authentication for access control using HTTP basic authentication
`mod_auth_digest.so`	auth_digest_module	Similar to auth_basic_module but instead of using a plain text authentication scheme, it uses a cryptographic one.
`mod_authn_alias.so`	authn_alias_module	Allows extended authentication providers to be created within the configuration file and assigned an alias name.
`mod_authn_anon.so`	authn_anon_module	Allows anonymous user access and logs the password given. Previously known asauth_anon_module.
`mod_authn_dbd.so`	authn_dbd_module	Allows provides authentication front-ends such as auth_digest_module and auth_basic_module to authenticate users by looking up users in SQL tables.
`mod_authn_dbm.so`	authn_dbm_module	Allows authentication front-ends such as auth_digest_module and auth_basic_module to authenticate users by looking up users in dbm password files. Previously known asauth_dbm_module.
`mod_authn_default.so`	authn_default_module	Fallback authentication module - it simply rejects any credentials supplied by the user.

Library	Module name	Description
mod_authn_file.so	authn_file_module	Allows authentication front-ends such as auth_digest_module and auth_basic_module to authenticate users by looking up users in plain text password files. This function was previously part of auth_module and auth_digest_module.
mod_authnz_ldap.so	authnz_ldap_module	Allows authentication front-ends such as auth_basic_module to authenticate users through an ldap directory. Previously known asauth_ldap_module.
mod_authz_dbm.so	authz_dbm_module	Group authorization using DBM files. This function was previously part of auth_dbm_module.
mod_authz_default.so	authz_default_module	Fallback authorisation module - it simply rejects any authorization request.
mod_authz_groupfile.so	authz_groupfile_module	Group authorization using plaintext files. This function was previously part of auth_module.
mod_authz_host.so	authz_host_module	Access control by browser hostname. Previously known asaccess_module.
mod_authz_owner.so	authz_owner_module	Authorization based on file ownership.
mod_authz_user.so	authz_user_module	Provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the web site. This function was previously part of auth_module.
mod_autoindex.so	autoindex_module	Automatically generates directory listings.
mod_cache.so mod_disk_cache.so mod_mem_cache.so	cache_module disk_cache_module mem_cache_module	Implements an RFC 2616 compliant HTTP content cache that can be used to cache either local or proxied content.
mod_cern_meta.so	cern_meta_module	An out-of-date way to specify some headers.
mod_cgi.so	cgi_module	Run CGI programs.
mod_cgid.so	cgid_module	Run CGI programs using an external CGI daemon.
mod_charset_lite.so	charset_lite_module	Specify character set translation or recoding.

Library	Module name	Description
mod_dav.so mod_dav_fs.so mod_dav_lock.so	dav_module dav_fs_module dav_lock_module	Distributed Authoring and Versioning functionality. (A standard for remote authoring and uploading.)
mod_dbd.so	dbd_module	Manages SQL database connections.
mod_deflate.so	deflate_module	Compress content prior to serving it.
mod_dir.so	dir_module	Supports the use of index.html files for directory lookups.
mod_dumpio.so	dumpio_module	Dumps all I/O to error log as desired..
mod_env.so	env_module	Changes the environment that CGI program are run in.
mod_expires.so	expires_module	Autogenerates the Expires: header according to user rules.
mod_ext_filter.so	ext_filter_module	Pass the response body through an external program before delivery to the client
mod_file_cache.so	file_cache_module	Caches a static list of files in memory
mod_filter.so	filter_module	Context-sensitive smart filter configuration module
mod_headers.so	headers_module	More general control of HTTP headers.
mod_imagemap.so	imagemap_module	Server-side image maps. Previously known as imap_module
mod_include.so	include_module	Server-side includes.
mod_info.so	info_module	Lets the server report on its configuration via a web request.
mod_ldap.so	ldap_module	LDAP connection pooling and result caching services for use by other LDAP modules.
mod_log_config.so	log_config_module	Configurable logging of requests and reponses.
mod_log_forensic.so	log_forensic_module	Forensic Logging of the requests made to the server.
mod_logio.so	logio_module	Logging of input and output bytes per request.
mod_mime.so	mime_module	Determines MIME types based on file names.
mod_mime_magic.so	mime_magic_module	Determines MIME types based on file contents.
mod_negotiation.so	negotiation_module	Provides for content negotiation between server and client.

Library	Module name	Description
mod_proxy.so	proxy_module	Lets your web server be a proxy. Typically it needs additional modules for specific protocols.
mod_proxy_ajp.so	proxy_ajp_module	AJP support module for proxy_module.
mod_proxy_balancer.so	proxy_balancer_module	mod_proxy extension for load balancing.
mod_proxy_connect.so	proxy_connect_module	Lets a proxying server handle CONNECT requests.
mod_proxy_ftp.so	proxy_ftp_module	Lets a proxying server handle FTP queries.
mod_proxy_http.so	proxy_http_module	Lets a proxying server handle HTTP queries.
mod_rewrite.so	rewrite_module	Allows for very complex rewriting of URLs before responding to them.
mod_setenvif.so	setenvif_module	Sets the environment for CGI programs based on properties of the request.
mod_speling.so	speling_module	Attempts to correct misspelled URLs.
mod_ssl.so	ssl_module	Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols.
mod_status.so	status_module	Provides information about the server's current status via a web request.
mod_suexec.so	suexec_module	Allows CGI scripts to run as a user other than wwwrun.
mod_unique_id.so	unique_id_module	Provides a unique key in the environment for each request.
mod_userdir.so	userdir_module	Allows for user's to have web pages from their home directories.
mod_usertrack.so	usertrack_module	Provision of cookies.
mod_version.so	version_module	Version dependent configuration.
mod_vhost_alias.so	vhost_alias_module	Allows for handling enormous numbers of virtual hosts without having to change the configuration each time.

A number of other modules are available in other SLES packages that depend on the apache2 package. Typically the package is named after the library. These are not supported or maintained by the Apache group. The truly brave may care to wander through the SLES "contributed" package sets for packages of Apache modules that aren't provided by Fedora at all. *caveat administrator.*

Table A-2. Modules shipped as part of other packages.

Package	Library	Module name
mod_auth_kerb	`mod_auth_kerb.so`	kerb_auth_module
mod_auth_mysql	`mod_auth_mysql.so`	mysql_auth_module
mod_auth_pgsql	`mod_auth_pgsql.so`	auth_pgsql_module
mod_authz_svn	`mod_authz_svn.so`	authz_svn_module
mod_dav_svn	`mod_dav_svn.so`	dav_svn_module
mod_perl	`mod_perl.so`	perl_module
mod_python	`mod_python.so`	python_module
php	`libphp4.so`	php4_module

Appendix B. Reference information for logging

Table B-1. Escape sequences for custom logs

%%	How to get "`%`" in the log line. Why would you want to?
%a	Client IP address
%A	Server IP address. Recall that you may be running different virtual hosts on different IP addresses.
%B	Number of data bytes sent back. (i.e. excluding headers)
%b	As for %B except that if the number is 0 then "`-`" is inserted instead.
%{*`fubar`*}C	The value of cookie *`fubar`*.
%D	The number of microseconds it took to serve the query. See %T below for a less accurate representation.
%{*`fubar`*}e	The value of environment variable *`fubar`* when the query was processed.
%f	The name of the file whose contents were ultimately served back to the client.
%H	The request protocol. (Typically HTTP or HTTPS.)
%{*`fubar`*}i	Value of the *`fubar`* header on the input query. See also %o below.
%l	The remote userid, if provided by RFCnnnn.
%m	The request method. Typically "GET" for our queries, but occasionally "HEAD" if the browser is smart. It may be "POST" for some CGI programs uploading data.
%{*`fubar`*}n	A record of a "note" passed from one module to another. Not of interest at our level.
%{*`fubar`*}o	The value of header *`fubar`* in the outgoing response headers. See also %i above.
%p	The port number of the server. Typically 80.
%P	The process ID of the child that serviced the query. Typically only of use for debugging and trouble-shooting.
%q	The query string component of the URL.
%r	The first line of the query.
%>s	The status code passed back to the client.
%t	The time of the request in standard format.
%{*`format`*}t	The time of the query in the format specified. See the manual page for `strftime` for details of the format.
%T	The time taken to service the query in seconds. See %D above for more accuracy.
%u	The userid used to authenticate to this page, if necessary.
%U	The URL requested without the server name and protocol elements and without any trailing query string.
%v	The server name for the virtual host that was given the query.

HTTP (*RFC 2616* (http://www.w3.org/Protocols/rfc2616/rfc2616.html)) is a very subtle protocol with much more happening than you might expect from the simple stuff we have been covering. The following table lists *all* the status codes it has and which might find themselves in your log files. In practise you will only see a tiny subset of them.

Table B-2. HTTP status codes

100	Continue
101	Switching protocols
200	OK
201	Created
202	Accepted
203	Nonauthoritative information
204	No content
205	Reset content
206	Partial content
300	Multiple choices
301	Moved permanently
302	Found
303	See other
304	Not modified
305	Use proxy
307	Temporary redirect
400	Bad request
401	Unauthorized
402	Payment required
403	Forbidden
404	Not found
405	Method not allowed
406	Not acceptable
407	Proxy authentication required
408	Request time-out
409	Conflict
410	Gone
411	Length required
412	Precondition failed
413	Request entity too large
414	Request URI too large
415	Unsupported media type
416	Requested range not satisfiable
417	Expectation failed
500	Internal server error
501	Not implemented
502	Bad gateway
503	Service unavailable
504	Gateway timed out

505	HTTP version unsupported