Web Server Management: Running Apache 2 on Red Hat Linux

Bob Dowling
University of Cambridge Computing Service

rjd4@cam.ac.uk

Web Server Management: Running Apache 2 on Red Hat Linux
by Bob Dowling

Installation: This course will first illustrate how to load the Apache 2 package on a Red Hat Linux version 9 system. This will be specific to this operating system.

ConfigurationThe 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.

Other utilities: The course will also discuss the associated utilities for log file rotation and reporting.

Table of Contents

Chapter 1. Installing the software

Packages:

We will describe the packages that Red Hat install if you request the *Web Server* group at installation. We will select the minimal set we need.

Running rpm:

We will demonstrate the use of the **rpm** utility to install these packages.

Changes to system:

We will briefly review what changes have been made to the system.

The web server group of packages

At installation the administrator is offered the choice of software to install. If the installer opted for the *Server* installation option or selected *Custom* and selected the *Web Server* group then all the necessary software is already installed.

Packages included in the *Web Server* group

httpd

The core web server package. Without this, we don't have a web server.

httpd-manual

This is the on-line manual for the web server.

mod_python

This allows the use of Python programs alongside the web server in a manner similar to, but more versatile than, classic CGI programs.

mod_perl

This is exactly the same as mod_python except that it supports the Perl programming language.

mod_ssl

This provides support for HTTPS, the encrypted version of HTTP. This is covered in a different Computing Service course (*Web Server Management: Securing Access to Web Servers*).

hwcrypto

This provides the facility to use hardware cryptographic systems to boost the speed of HTTPS communications.

php

PHP provides another scripting language for the web server to replace CGI.

php-imap

This package provides the hooks for PHP to use the IMAP protocol to talk to email systems.

php-ldap

This package provides the hooks for `PHP` to perform lookups in LDAP directories.

php-pgsql

This package provides the hooks for `PHP` to work with `PostgreSQL` relational databases.

mod_auth_pgsql

This package provides the facility to use a `PostgreSQL` database to do password lookups for access controls rather than the plain text files that are often used.

squid

`Squid` is a proxy caching server.

tux

`Tux` is an alternative web server. We will not be discussing `tux` in this course.

webalizer

`Webalizer` is a Web server log analysis program. We will examine its use in the Section called *Log file analysis* in Chapter 7.

There are other packages released with Red Hat Linux version 9 which depend on the httpd package and which enhance its functionality. In particular, if you want to connect to a database running `MySQL` rather than `PostgreSQL` then the corresponding packages are mod_auth_mysql and php-mysql.

For the purposes of this course we will need only some of the packages. We will also make use of another system package for log rotation. This is a base system package and should already be installed on the system.

Packages used directly in this course

- httpd
- webalizer
- logrotate

Installing the packages

Location:

We detail where to find the packages. This element is Cambridge-specific.

rpm:

Then we describe exactly how to install the packages.

Components

Finally we review what's actually been added to the system (before we delete large chunks of it).

To install the packages 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
#
```

Figure 1-1. Changing to be root

Unix Support keeps the Red Hat distributions on an NFS server nfs-uxsup.csx.cam.ac.uk in the directory `/linux/redhat`. To access it we will create a directory to mount the server's directory tree on, mount it, install the packages and then unmount the distribution again.

```
# mkdir /mnt/redhat
# mount  -o ro  nfs-uxsup.csx.cam.ac.uk:/linux/redhat  /mnt/redhat
# cd /mnt/redhat
# ls
5.2  7.1  8.0   code     enterprise    rawhide
6.2  7.2  9     contrib  local_extras  updates
7.0  7.3  beta  current  preview
```

Figure 1-2. Mounting the NFS server's Red Hat Linux distributions

Next, we will move to the directory that has the packages for the specific version of Red Hat Linux that we run. Red Hat have distributions for various different languages and hardware types. Each of these is available at various different versions. We start by identifying these to navigate to a directory which is the contents of the corresponding Red Hat CD.

Locating the CD image

- Version: We are using version *9*.
- Language: We want the English language version, identified by code *en*. Actually, Unix Support only mirrors the English versions.
- Product: Red Hat distribute a variety of products including the operating system. They also ship documentation, extra utilities etc. We want the operating system product, identified by *os*.
- Architecture: We want the Intel architecture, identified by code *i386*.

```
# pwd
/mnt/redhat
# cd 9/en/os/i386
# ls
autorun                README.zh_CN
dosutils               README.zh_TW
EULA                   RedHat
GPL                    RELEASE-NOTES
images                 RELEASE-NOTES-de.html
isolinux               RELEASE-NOTES-es.html
jpk                    RELEASE-NOTES-fr.html
README                 RELEASE-NOTES.html
README-Accessibility   RELEASE-NOTES-it.html
README.de              RELEASE-NOTES-ja.html
README.es              RELEASE-NOTES-ko.html
README.fr              RELEASE-NOTES-zh_CN.html
README.it              RELEASE-NOTES-zh_TW.html
README.ja              RPM-GPG-KEY
README.ko              SRPMS
```

Figure 1-3. Navigating to the CD image

This has brought us to the copy of the relevant Red Hat Linux CD. Now we have to locate the packages. These can be found in the subdirectory `RedHat/RPMS`.

```
# cd RedHat/RPMS
# ls
4Suite-0.11.1-10.i386.rpm
a2ps-4.13b-24.i386.rpm
abiword-1.0.2-6.i386.rpm
 ...
zlib-1.1.4-4.i386.rpm
zlib-devel-1.1.4-4.i386.rpm
zsh-4.0.4-8.i386.rpm
```

Figure 1-4. Navigating to the package repository

Each of these `.rpm` files contains a Red Hat package. We are now in a position to install the ones we want. Each has a file name based on the package name. We will install the software with the **rpm** command's `--install` option. These filenames tend to be quite long. **Tab**-completion can save you a lot of typing. Type the beginning of a filename and press **Tab**. The shell will complete the filename as far as it can.

```
# pwd
/mnt/redhat/9/en/os/i386/RedHat/RPMS
# rpm --install httpd-2.0.40-8.i386.rpm
# rpm --install webalizer-2.01_10-9.i386.rpm
```

Figure 1-5. Installing the packages

The two filenames could have been given as a list on a single **rpm** command line.

We have been slightly unfair here. We have installed the version of the web server that comes with Red Hat Linux version 9. However, there may have been releases since then that we should upgrade to. You may recall an `updates` directory in `/mnt/redhat`. This contains updates to software that are released after the initial release.

We will use the **rpm** command's `--freshen` option to update our version of the core httpd package.

```
# cd /mnt/redhat/updates/9/en/os/i386
# ls httpd-* logrotate-* webalizer-*
ls: logrotate-*: No such file or directory
ls: webalizer-*: No such file or directory
httpd-2.0.40-11.i386.rpm        httpd-manual-2.0.40-11.i386.rpm
httpd-devel-2.0.40-11.i386.rpm
# rpm --freshen httpd-2.0.40-11.i386.rpm
```

Figure 1-6. Looking for updated packages

Finally we will unmount the NFS file server. To do this we have to leave any of the directories we have mounted from it as we can't whip the carpet out from under our own feet.

```
# pwd
/mnt/redhat/9/en/os/i386/RedHat/RPMS
# cd
# pwd
/root
# umount /mnt/redhat
```

Figure 1-7. Disconnecting from the file server

Changes made to the system

We should quickly examine what changes have been made to the system by the installation of these packages. The httpd package has obviously installed a web server (in `/usr/sbin/httpd`) but it has also installed files in a few other locations.

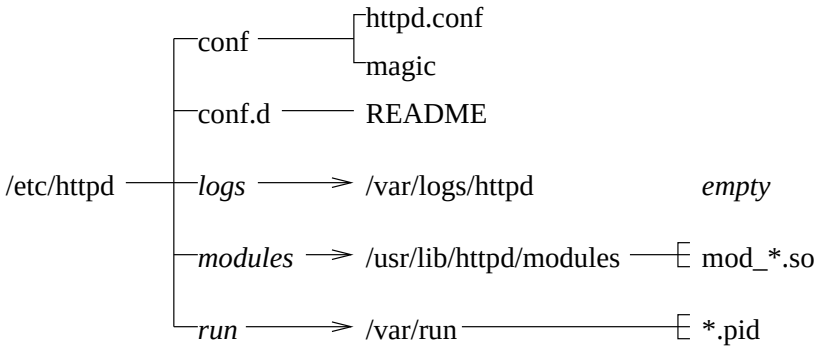

Figure 1-8. Configuration hierarchy

The `/etc/httpd` directory contains all the configuration for the web server and is where the server looks for everything except the web pages themselves. The `conf` subdirectory is for the main configuration file, `httpd.conf`. The `conf.d` directory is for extra elements of configuration which can be automatically included in the server configuration.

Because this directory is where the server looks for most things there are some symbolic links leaving this directory and pointing to where the *system* expects such things to go.

The `logs` symbolic link points into the `/var/logs` directory where Red Hat Linux stores its logs. An `httpd` 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.

The `modules` symbolic link points into `/usr/lib` where Red Hat Linux keeps its libraries. Again, an `httpd` subdirectory is used to keep all the Apache libraries together. The libraries are called "modules" under Apache and much of the course will be devoted to what they can do and how to get them to do it.

The `run` symbolic link points to `/var/run`. This directory on a Red Hat Linux system is used by long-running processes to declare that they are running and to indicate how to contact them. In practice this means that it is full of *PID files*. These are files, named after the process they refer to, containing the process ID (PID) of the process. For Apache, the PID file is called `httpd.pid`. In practice, we will not need to know about this file except to understand how the Apache shutdown procedure works in Chapter 3.

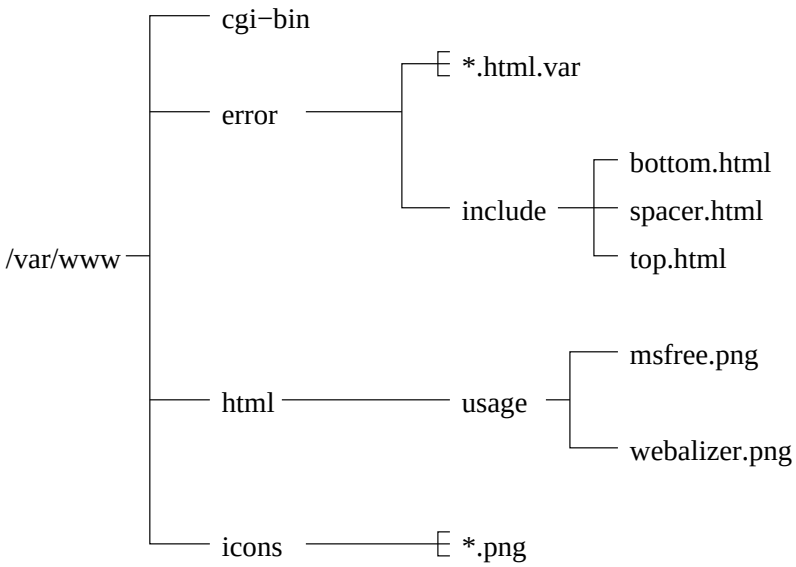

Figure 1-9. Initial document hierarchy

The `/var/www` directory is the default location for files served by the web server. One of its subdirectories, `cgi-bin` is empty as Apache does not ship with CGI scripts any more. This course does not cover CGI programming. There is a different University Computing Service course, *CGI Scripting for Programmers: Introduction*, for this.

The `html` subdirectory is similarly almost empty. Its only content is a directory `usage` which has nothing to do with how to run the web server but is, rather, where the `webalizer` log analysis application puts its reports on how much the server has been used. We will cover this in depth in the Section called *Log file analysis* in Chapter 7. The `html` directory 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.

The `icons` subdirectory contains the icons used in the automatically generated listings. We will use these extensively when we discuss automatic indexing of directories in the Section called *Automatic indexing of directories* in Chapter 6.

The most complex installed directory structure is in `error`. This contains a large number of files whose names are HTTP error names with `.html.var` as a suffix. Each of these contains the web page that will be displayed if the corresponding error condition happens. We will not be discussing error documents in this course. Each error page has the error in four different languages and the web server is geared up to serve the appropriate language from the choice. (This is what the `.var` suffix is all about.)

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

Two other changes have been made to the system. A user and group have been created for the web server to run as.

```
$ grep apache /etc/passwd /etc/group
/etc/passwd:apache:x:48:48:Apache:/var/www:/sbin/nologin
/etc/group:apache:x:48:
```

Figure 1-10. The apache user and group

User programs included with the web server

- `/usr/bin/ab`: This a stress-tester for the web server. Please do not stress-test people's servers without their explicit permission. Otherwise you may find them stress-testing your skull with a pickaxe handle.
- `/usr/bin/htdbm`, `/usr/bin/htdigest`, `/usr/bin/htpasswd`: These manipulate user and password information for web access controls. We will see **htpasswd** 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/bin/logresolve`: 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 hostnames 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 hostnames) in Chapter 7.

System programs included with the web server

- `/usr/sbin/apachectl`: 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/httpd`: This is the web server itself.
- `/usr/sbin/rotatelogs`: This is an Apache utility providing for rotating log files. However, it is not used by a Red Hat Linux 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/suexec`: 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.

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

Figure 1-11. Enabling the web server

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.

```
# cd
# /etc/init.d/httpd start
Starting httpd:                    [  OK  ]
```

Figure 1-12. Manually starting the web server

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

What you get for the quick & dirty approach

- If the server's DNS name is *server* then any file placed in `/var/www/html/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/httpd` called `access_log` and `error_log`. These will be rotated weekly and four weeks' worth of logs will be kept.

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 `/var/www/html`. We will create a group, webadmin, who will have access to the site.

Setting up and using the webadmin group

1. Creating the group

   ```
   # groupadd -r webadmin
   ```

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

2. Setting up `/var/www/html`

   Next we have to change `/var/www/html` so that this newly created group has sway over it. We need to do a number of things.

   - We must change the group of the directory to be webadmin. 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.

   ```
   # chgrp webadmin /var/www/html
   # chmod g+ws /var/www/html
   ```

3. Adding users to the group

   We will add the users alice and bob to the webadmin 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.

   ```
   apache:x:48:
   webalizer:x:67:
   webadmin:x:101:alice,bob
   ```

   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 webadmin we must state that she is in webadmin and alpha.

```
# usermod -G alpha,webadmin alice
```

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

```
# usermod -G webadmin bob
```

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

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 and we will describe the two most common.

The server we want

We are going to describe the server in the terms that an academic would use to describe it to a computer officer in the University: annoyingly vague. This is an excuse for this course to introduce new features one at a time, of course.

Sites:

We are told that the server must serve two web sites, chalk.dept.cam.ac.uk and cheese.dept.cam.ac.uk whose 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 academic or 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 and usage information should be available on the web. We will have to consider the DPA implications of this part of the specification, of course.

What is a virtual host?

There are (at least) three different ways that a single web server can host more than one web site.

Multiple ports:

A web server can listen on more than the default 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/. The receiving system uses the port number of the incoming query to distinguish between web sites.

Multiple addresses:

A single system need not have a single IP address. It can have many and each can have a different web site attached to it. This leads to two different server names appearing in standard URLs (i.e. there's no :*port* element in the URL) but the two server names correspond to the two different IP addresses of the system and correspond to different web sites. The receiving web server uses the destination IP address on the incoming query to distinguish web sites.

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 two different names in the DNS correspond to that address. So chalk.dept.cam.ac.uk and cheese.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?)

```
GET /index.html HTTP/1.1
Host: noether.csi.cam.ac.uk
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.1) Gecko/20021003
Accept: text/html,text/plain;q=0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en, es;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.33
Connection: keep-alive
Keep-Alive: 300
```

Figure 2-1. Incoming HTTP query for http://noether.csi.cam.ac.uk/index.html

To understand name-based virtual hosting consider just the first two lines. The **GET** request only refers to the local element of the URL. The second line specifies the hostname 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: noether.csi.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-US; rv:1.0.1) Gecko/20021003`

This optional line identifies the browser. Some servers, designed by stupid people, vary the output according to this header. 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. `X11` indicates that it is a browser with access to an X windowing system, `U` identifies it as a Unix and `Linux i686` specifies what sort of Unix. `en-US` specifies the locale the browser is working in. `rc:1.0.1` specifies the version of the browser. `Gecko` specifies the rendering engine the browser uses and `20021003` is the release date (as version number) of the version.

`Accept: text/html,text/plain;q=0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1`

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/html means that the browser is happy to accept MIME content type text/html and text/plain means that it can accept plain text too. The qualifier q=0.8 means that, given a choice, the browser would prefer to receive text/html (default q=1.0) than text/plain (q=0.8). Similarly for images, the browser would prefer image/png or image/jpeg to the image/gif format. Finally it will accept any format (*/*) but is not keen on them (q=0.1).

We will meet MIME content types again in Chapter 4.

`Accept-Language: en, es;q=0.50`

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. The author is learning Spanish and has Spanish as a second choice in the language selections.

`Accept-Encoding: gzip, deflate, compress;q=0.9`

Just as there was negotiation over MIME content type there can also be negotation 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=0.66, *;q=0.33`

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 a second choice and everything else coming third.

`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 fficiency 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, I'll also post the response. To make the example work, I've installed a trivial `index.html` web page. We will use this later

```
HTTP/1.1 200 OK
Date: Wed, 19 Mar 2003 09:41:23 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 19 Mar 2003 09:40:21 GMT
ETag: "e011-13e-12910b40"
Accept-Ranges: bytes
Content-Length: 318
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<title>The DEFAULT web site</title>
</head><body>
<h1>Welcome to DEFAULT</h1>
<p>This is the DEFAULT web site.</p>
</body>
</html>
```

Figure 2-2. Outgoing HTTP response from the server

Chapter 3. Getting started

Clean slate:

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.

Two more lines

We will add two more lines to the most basic configuration file we can have. The first will just help the configuration file make more sense and the second will turn off many defaults so we can see them explicitly when we need them.

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/httpd/conf/httpd.conf
rm: remove regular file '/etc/httpd/conf/httpd.conf'? y
```

Figure 3-1. Deleting the configuration file

The web server will not start now. First it will complain about not having a configuration file.

```
# /etc/init.d/httpd start
Starting httpd: grep: /etc/httpd/conf/httpd.conf: No such file or directory
httpd: could not open document config file /etc/httpd/conf/httpd.conf
                                                           [FAILED]
```

Figure 3-2. Failing to start the server with no configuration

Close observers will notice that it complains twice. The first is an error from the **grep** command. This is just a Red Hat Linux feature that we can ignore. (The Red Hat Linux start script checks to make sure that you aren't using an Apache 1 configuration file.) The second error message comes from **httpd** and is the one we are interested in.

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

```
# touch /etc/httpd/conf/httpd.conf
# /etc/init.d/httpd start
Starting httpd: no listening sockets available, shutting down
                                                           [FAILED]
```

Figure 3-3. Failing to start the server with an empty configuration

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
```

Figure 3-4. `httpd.conf`: The Listen command

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 is all we need!

```
# /etc/init.d/httpd start
Starting httpd:                                            [  OK  ]
```

Figure 3-5. Launching the web server with a minimal configuration

Unfortunately, the launched web server then immediately shuts down. By default, the web server will log error messages in `/var/log/httpd/error_log`. We can look in there for clues as to what we need next.

```
[Mon Mar 17 17:51:13 2003] [notice] Apache/2.0.40 (Red Hat Linux)
configured -- resuming normal operations
[Mon Mar 17 17:51:13 2003] [alert] (2)No such file or directory:
getpwuid: couldn't determine user name from uid 4294967295, you
probably need to modify the User directive
 ...
[Mon Mar 17 17:51:14 2003] [alert] Child 9315 returned a Fatal error...
Apache is exiting!
```

Figure 3-6. `error_log`: Launch and fail

What does this error message mean? It means that the web server needs to know who to run as. You will recall that the software installation created a user and group for the server to run as. We need to tell it to use them. This is done with the **User** and **Group** commands in the configuration file.

```
User    apache
Group   apache
```

Figure 3-7. `httpd.conf`: User and group

We have cheated. There is one configuration line we have omitted because in the Red Hat Linux build of Apache there is an appropriate, compiled-in default. A number of files will be referred to in this configuration file (such as the modules we will start to meet in Chapter 4 and the log files we will meet in Chapter 7) and we want to give relative path names for these files. We need to specify one absolute path for these to be relative to. We add the **ServerRoot** command to identify this directory for completeness.

The second "unnecessary" line we will add 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    apache
Group   apache
ServerRoot /etc/httpd
Options None
```

Figure 3-8. `httpd.conf`: Minimal version

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 Red Hat Linux Apache package.

Group

This specifies the system group ID that the server process should run as. This group was created by the Red Hat Linux Apache package.

ServerRoot

All unqualified filenames in the configuration file will be resolved with respect to this directory except for the actual web pages which will be handled by a different command.

Options

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

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

```
# /etc/init.d/httpd start
Starting httpd:                                            [  OK  ]
# tail -1 /var/log/httpd/error_log
[Mon Mar 17 18:02:56 2003] [notice] Apache/2.0.40 (Red Hat Linux) configured -- resuming norma
# ps -ef | grep httpd
root      9344     1  0 18:02 ?        00:00:00 /usr/sbin/httpd
apache    9345  9344  0 18:02 ?        00:00:00 /usr/sbin/httpd
apache    9346  9344  0 18:02 ?        00:00:00 /usr/sbin/httpd
apache    9349  9344  0 18:02 ?        00:00:00 /usr/sbin/httpd
apache    9350  9344  0 18:02 ?        00:00:00 /usr/sbin/httpd
apache    9351  9344  0 18:02 ?        00:00:00 /usr/sbin/httpd
root      9357  8826  0 18:05 pts/3    00:00:00 grep httpd
```

Figure 3-9. Successful launch

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 apache. 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 9344). It in turn launched five child processes owned by apache (PIDs 9345–9351).

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 other than pre-forked, but Red Hat Linux has this enabled by default and it is what we will use throughout the course.

Figure 3-10. No documents!

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/httpd/error_log` we will see the error message:

```
[Tue Mar 18 13:42:10 2003] [error] File does not exist: /etc/httpd/htdocs
```

which indicates that the server is looking in the wrong place!.

We need to tell it where to look. 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 chalk.dept.cam.ac.uk nor cheese.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, `CHALK` and `CHEESE`, of `/var/www` for the two web-sites. We will also create two groups, chalk and cheese, which will contain the people

entitled to update the sites.

```
# cd /var/www
# mkdir CHALK CHEESE
# groupadd -r chalk
# groupadd -r cheese
# chgrp chalk CHALK
# chgrp cheese CHEESE
# chmod g+ws CHALK
# chmod g+ws CHEESE
```

Figure 3-11. Setting up the two web sites

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 HTML 3.2 Final//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<title>The CHALK web site</title>
</head><body>
<h1>Welcome to CHALK</h1>
<p>This is the CHALK web site.</p>
</body>
</html>
```

Figure 3-12. The chalk `index.html` file

Now we must tell the web server to use these two directories appropriately.

```
NameVirtualHost *

<VirtualHost *>
ServerName      chalk.dept.cam.ac.uk
DocumentRoot    /var/www/CHALK
</VirtualHost>

<VirtualHost *>
ServerName      cheese.dept.cam.ac.uk
DocumentRoot    /var/www/CHEESE
</VirtualHost>
```

Figure 3-13. `httpd.conf`: Setting up the virtual hosts

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 chalk and one for cheese. 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>

The **VirtualHost** section describes a single virtual host. Everything from the **<VirtualHost *`interface`*>** to **</VirtualHost>** sets parameters for a single virtual host. 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 does not match this then the virtual host block will not 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.

We do not need to restart our web server after each change to the configuration file. A rather faster mechanism is to cause it to reread its file to note changes. This is done by using the `reload` option on the startup script.

```
# /etc/init.d/httpd reload
Reloading httpd:                                           [  OK  ]
```

Figure 3-14. Getting the server to reread its configuration file

We are now running one web server supporting two web sites. However, if we request the `index.html` page from chalk.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.

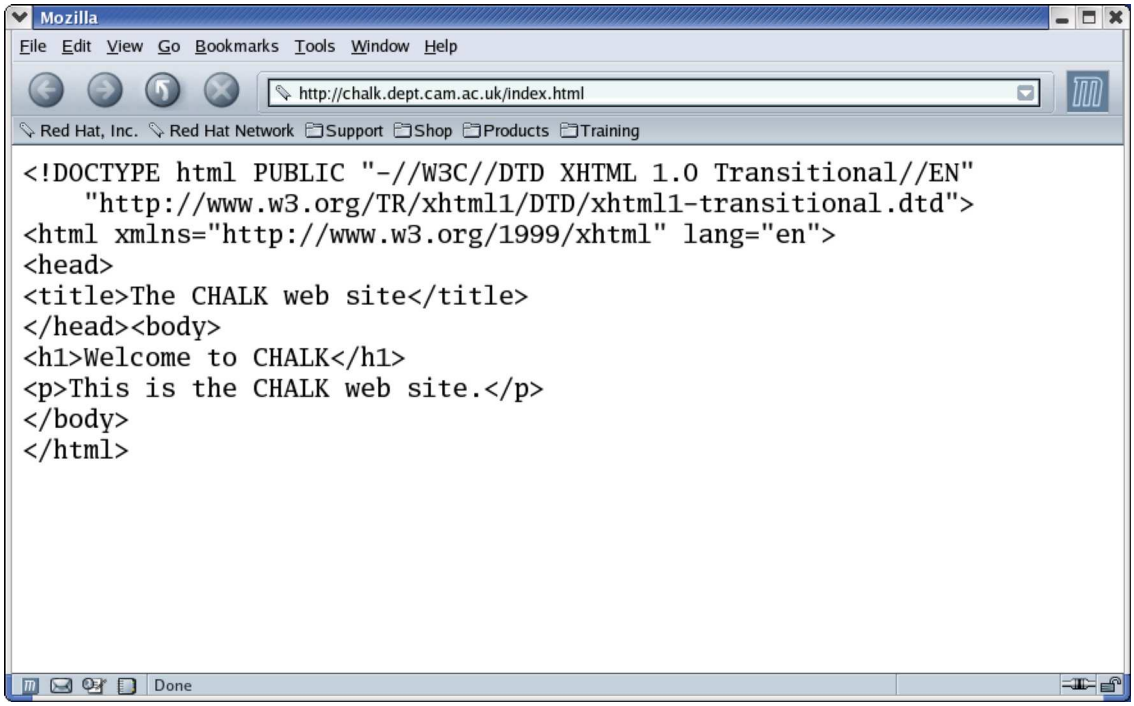

Figure 3-15. Contacting the chalk.dept.cam.ac.uk web site; `index.html`

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/httpd`

`start`

Starts the web server.

`stop`

Stops the web server.

`restart`

Stops and starts the web server.

`condrestart`

Stops and starts the web server if the PID file exists that suggests that the web server was started via this script, rather than just manually.

`status`

Indicates whether or not the web server is running.

`fullstatus`

This option does not run on Red Hat Linux.

`reload`

Causes a running web server to reread its configuration file(s) and to reopen its log files.

`graceful`

Equivalent to `restart` but politer.

`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.

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

```
HTTP/1.1 200 OK
Date: Wed, 19 Mar 2003 09:41:23 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 19 Mar 2003 09:40:21 GMT
ETag: "e011-13e-12910b40"
Accept-Ranges: bytes
Content-Length: 318
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

Figure 4-1. Outgoing HTTP response headers from a working server

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. It also tells the browser that the underlying character set used for the web page is ISO-8859-1.

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

```
HTTP/1.1 200 OK
Date: Wed, 19 Mar 2003 10:02:11 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Tue, 18 Mar 2003 14:22:06 GMT
ETag: "2813-138-e4577f80"
Accept-Ranges: bytes
Content-Length: 312
Connection: close
Content-Type: text/plain
```

Figure 4-2. Outgoing HTTP response headers from our server

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

MIME types on a Red Hat Linux 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
file: Using regular magic file '/usr/share/magic.mime'
course.ps: application/postscript
$ file -i course.ps 2> /dev/null
course.ps: application/postscript
$ file -i course.pdf 2> /dev/null
course.pdf: application/pdf
```

Figure 4-3. Using the file command

As you will deduce from the warning message printed on standard error, the file `/usr/share/magic.mime` is used to store the information about how to map from content to MIME type. The default file `/usr/share/magic` 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 base system (as part of the mailcap 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
```

Figure 4-4. Extracts from `/etc/mime.types`

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 the line to the configuration file that loads the module.

```
LoadModule      mime_module     modules/mod_mime.so
TypesConfig     /etc/mime.types
```

Figure 4-5. `httpd.conf`: Loading the MIME module

The **LoadModule** command takes *two* arguments. The second is the filename of the shared library that it needs. Note that the pathname of the module is given relative to the `/etc/httpd` directory. The first argument is the name of the module within that file.

Normally you would consult the documentation to determine what a module's name is but there is a filthy hack you can often use to determine it automatically.

```
$ nm /etc/httpd/modules/mod_mime.so | grep --word D
000031e0 D mime_module
```

Figure 4-6. Getting module names from shared libraries

Don't worry about what the line means in detail. If you really want to know then read the fine manual page for **nm**. The third element in the answer is the module name.

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.

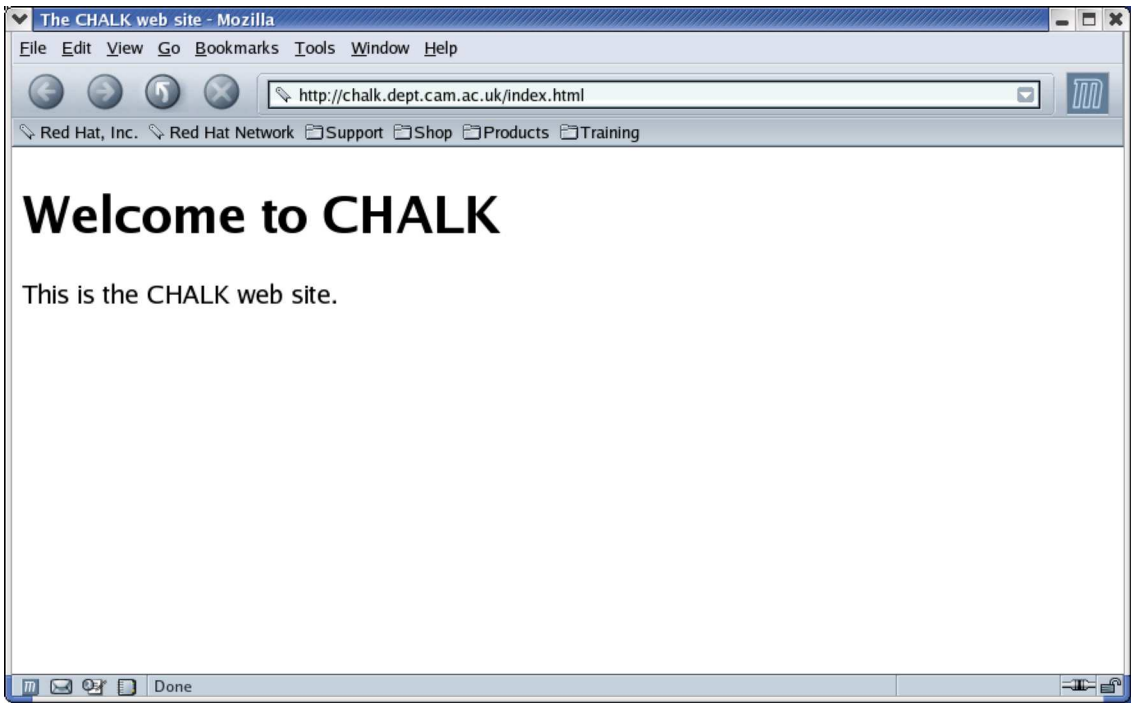

Figure 4-7. chalk.dept.cam.ac.uk index page as HTML

Syntax summary: Loading modules

LoadModule ***library.so name***

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 Red Hat Linux system this file is `/etc/mime.types` and is installed by the mailcap package.

Chapter 5. Symbolic links

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.

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 /var/www/CHALK/
$ ln -s index.html main.html
$ ls -l
total 4
-rw-rw-r--  1  rjd4 chalk  312 may  7 10:01 index.html
lrwxrwxrwx  1  rjd4 chalk   10 may  7 10:19 main.html -> index.html
```

Figure 5-1. Creating the symbiolic link `main.html`

We note that 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.

Figure 5-2. We are forbidden to access `main.html`

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
```

Figure 5-3. Setting a single option

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

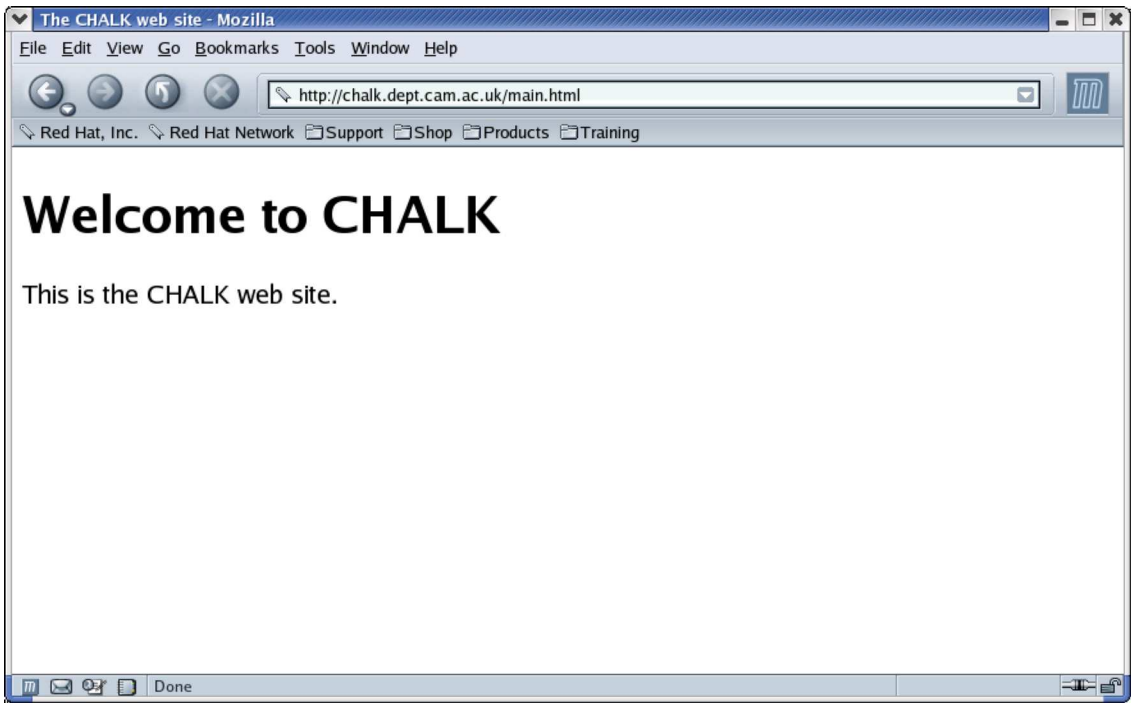

Figure 5-4. We are permitted to access `main.html`

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.

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.

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://cheese.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.

Figure 6-1. / causing problems

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      modules/mod_dir.so
DirectoryIndex  index.html index.htm
```

Figure 6-2. `httpd.conf`: Using dir_module

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.

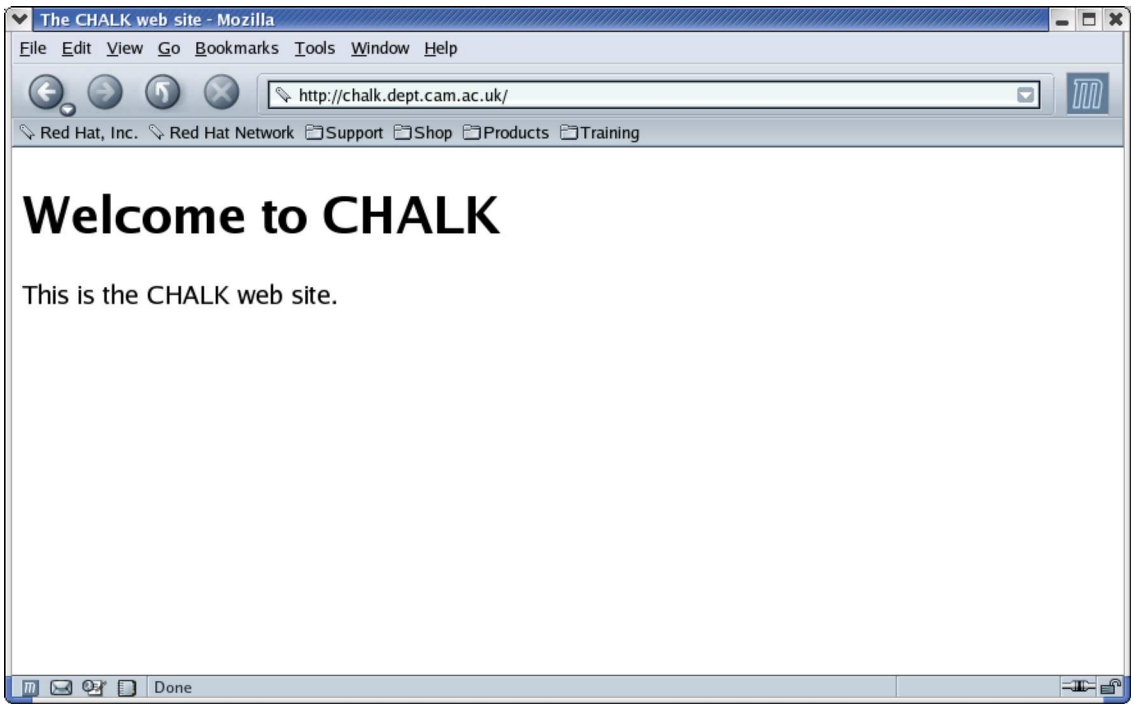

Figure 6-3. `/` equivalent to `/index.html`

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.

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 modules/mod_autoindex.so
```

Figure 6-4. `httpd.conf`: Loading autoindex_module

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. The web server now knows how to handle directories but has decided not to.

Figure 6-5. Refusal to handle a directory

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** lin in the configuration file.

```
LoadModule autoindex_module modules/mod_autoindex.so
Options         +Indexes
```

Figure 6-6. `httpd.conf`: Loading and enabling autoindex_module

And now, if we ask for the `/` URL we get the list of the single file (`index.html`) that appears in the top-level directory.

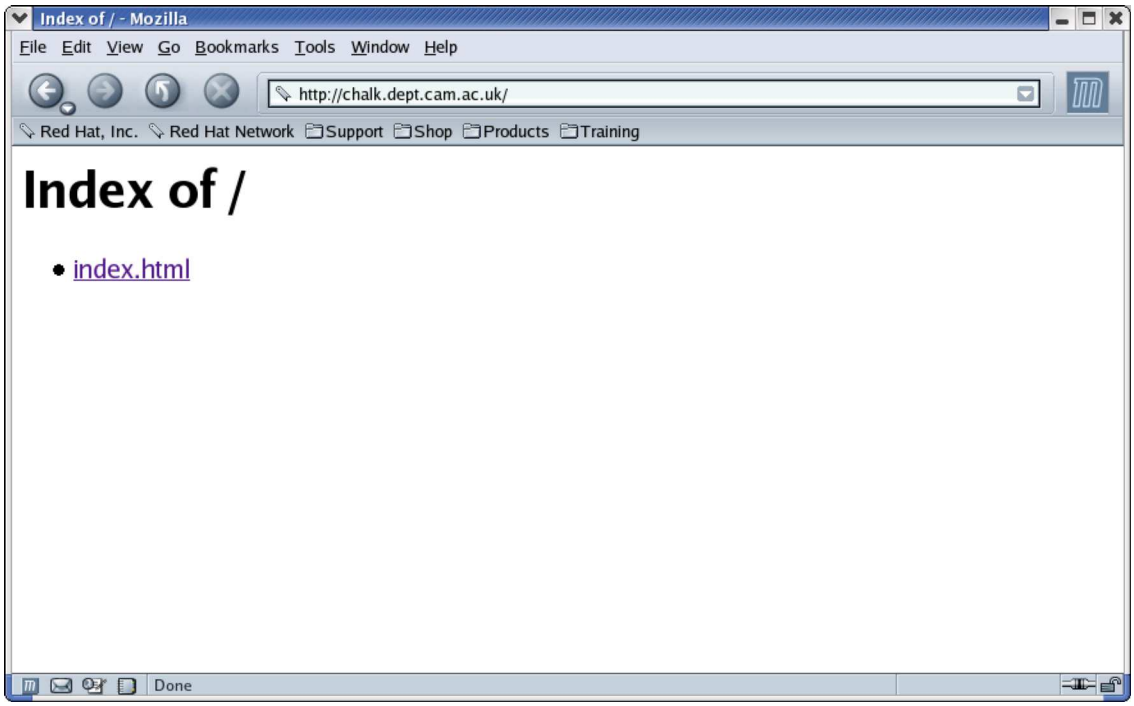

Figure 6-7. The top-level listing

By default, the index produced is a very simple one (an itemized list in HTML). The module provides a command, **IndexOptions**, which allows us to present a bit more information about the files.

To see it in operation we will add a simple **IndexOptions** command to our configuration file to turn on "fancy indexing".

```
IndexOptions FancyIndexing
```

Figure 6-8. `httpd.conf`: Turning on fancy indexing

Now when we ask for the `/` URL we get a different format of output.

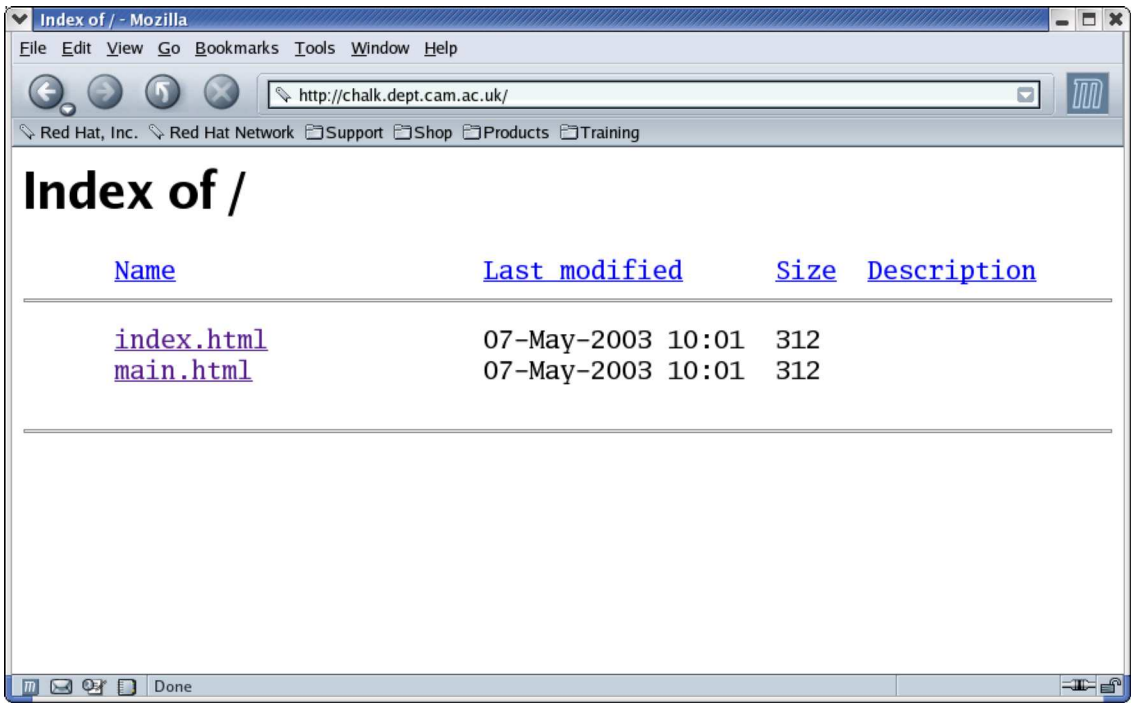

Figure 6-9. The fancy index of /

Columns in default fancy index

- Name
- Size
- Last modified
- Description

The layout of this page is very primitive. Arguably the presentation of this information should be an HTML table, but the auto-indexing module predates the widespread support for tables in web browsers so a plain text layout, limited to 80 columns wide, is used by default. We will show how to override this default below. Because the text is so limited in width, facilities are provided to select which columns are shown and there is limited support for changing the widths of columns. There are also controls on which *rows* get shown.

So we can investigate this module more closely, we will populate the cheese.dept.cam.ac.uk web site.

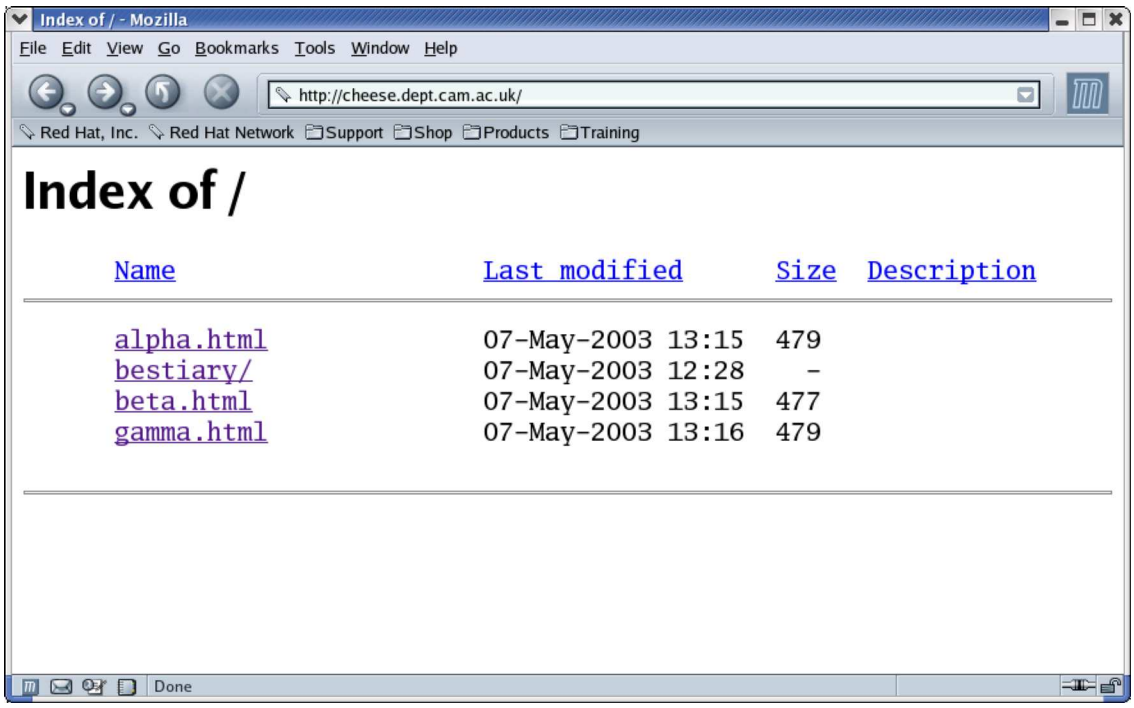

Figure 6-10. A populated cheese.dept.cam.ac.uk website

Manipulating columns

We will start by making some use of the Description column. There are mechanisms for manually adding descriptions to entries in the list, but these additions are usually delegated to the directory in question rather than done in the main configuration file. Therefore we will leave this topic until we cover delegation in Chapter 9. Instead, we will use an Apache facility to parse the HTML in web pages, extract the `<TITLE>` entries and uses these as the descriptions. This is done by adding another option to the **IndexOptions** command.

```
IndexOptions FancyIndexing ScanHTMLTitles
```

Figure 6-11. `httpd.conf`: Adding title-based descriptions

Now the right-hand column has content.

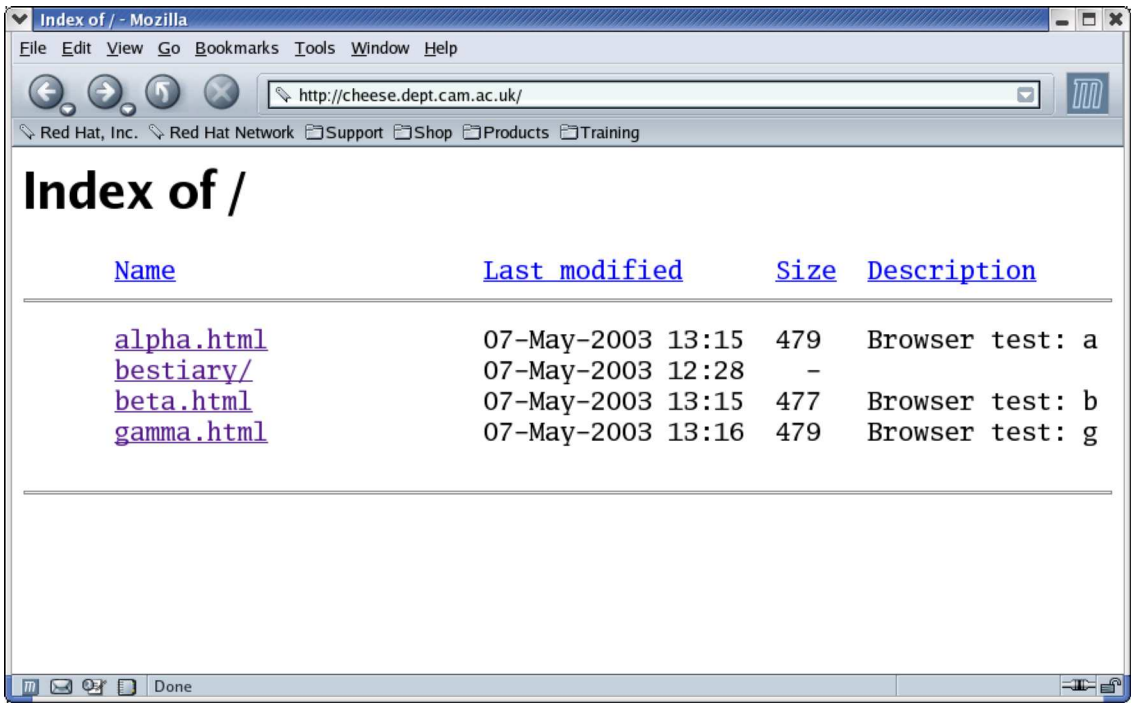

Figure 6-12. The cheese.dept.cam.ac.uk website with descriptions

Next, we will remove unwanted columns. For the sake or argument, suppose we don't want the Size or Last modified columns. These can be suppressed with two more indexing options.

```
IndexOptions    FancyIndexing ScanHTMLTitles SuppressSize SuppressLastModified
```

Figure 6-13. `httpd.conf`: Suppressing the Size and Last updated columns

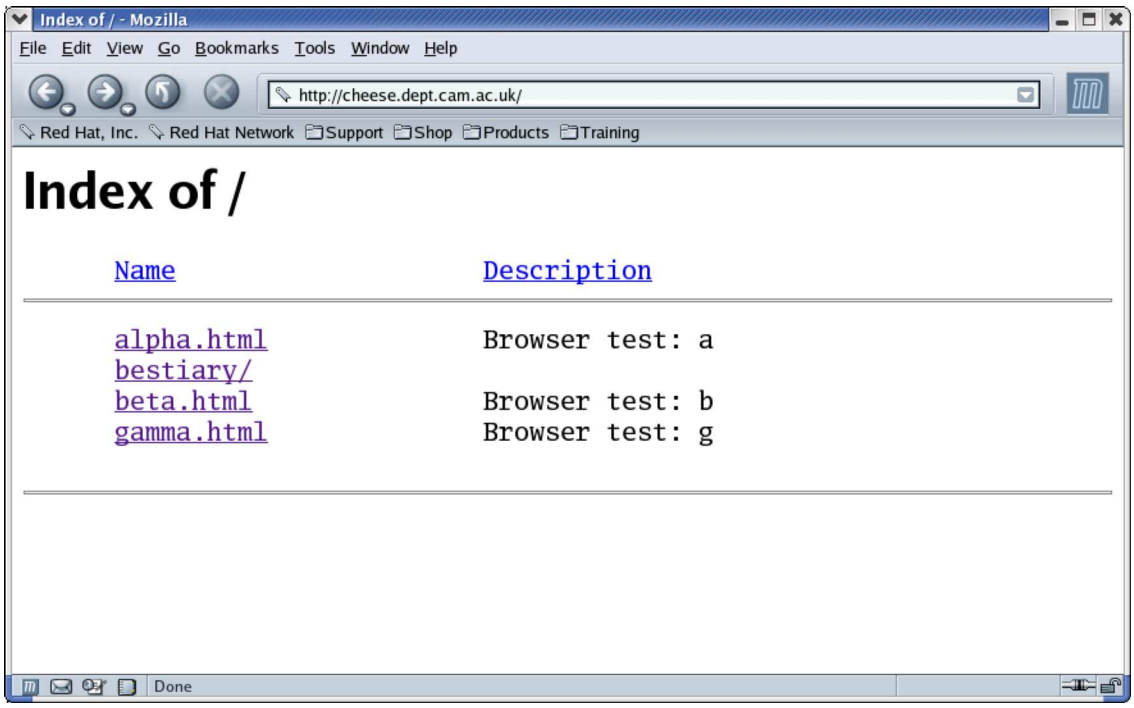

Figure 6-14. The displayed listing with columns suppressed

Finally, we will look at modifying the widths of the various columns we have. We will modify `beta.html` to have a very long title, pushing the Description column too far.

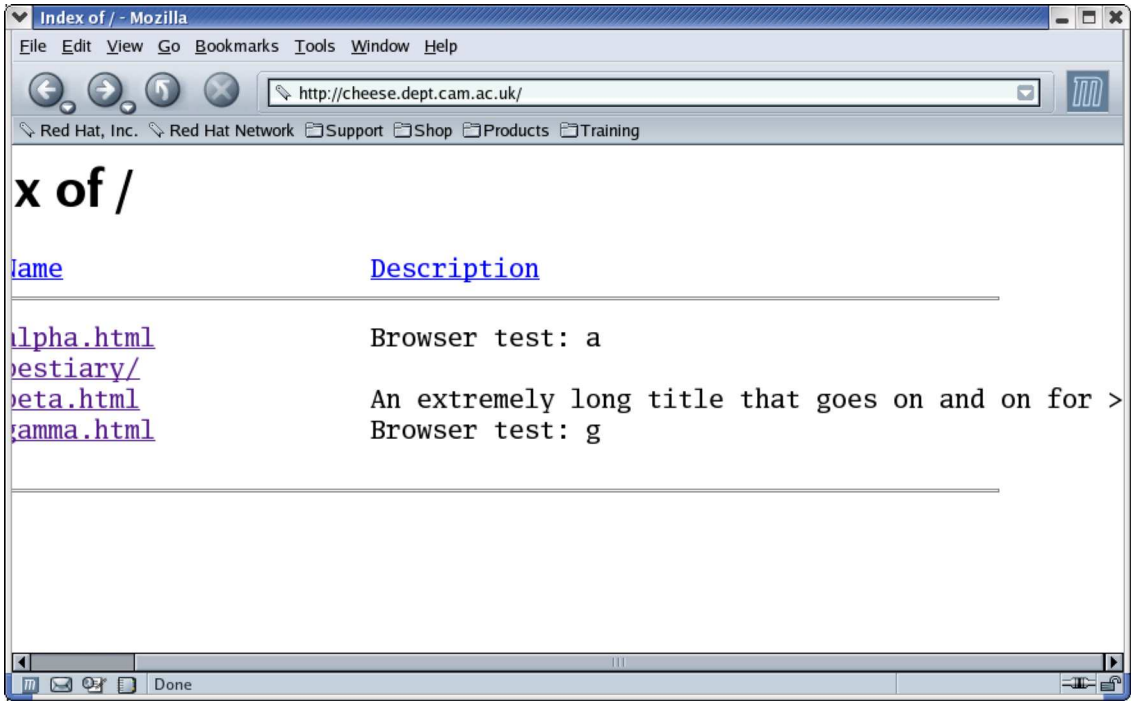

Figure 6-15. Too long a description

Because the module dates back to the very beginning, it is designed around the idea of the text-based browser with an 80-character width. The module truncates the de-

scription of the file if it would push the row beyond this point and indicates the truncation with a ">". There are two indexing options which can be used to alleviate this behaviour, `NameWidth` and `DescriptionWidth`. Each of these can be used to specify an absolute character width of the column or can be used to instruct the server to determine the longest item in the list and adjust accordingly. We will use this second approach.

```
IndexOptions  FancyIndexing ... NameWidth=* DescriptionWidth=* ...
```

Figure 6-16. `httpd.conf`: Auto-adjusting the column widths

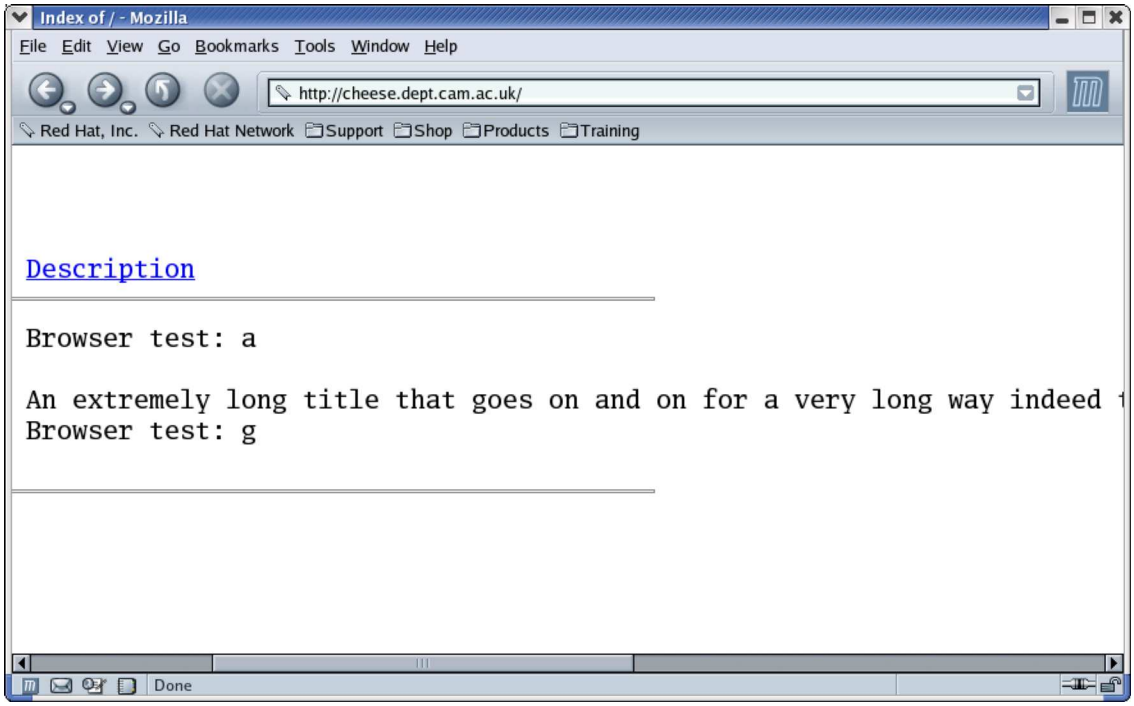

Figure 6-17. A very wide description column

Manipulating rows

The first thing we will do is to 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 (`delta.html`, say) creates work files (`#delta.html#`) while it is running and leaves behind backup files (`delta.html~`) when it is finished. We don't want these files appearing in the listings.

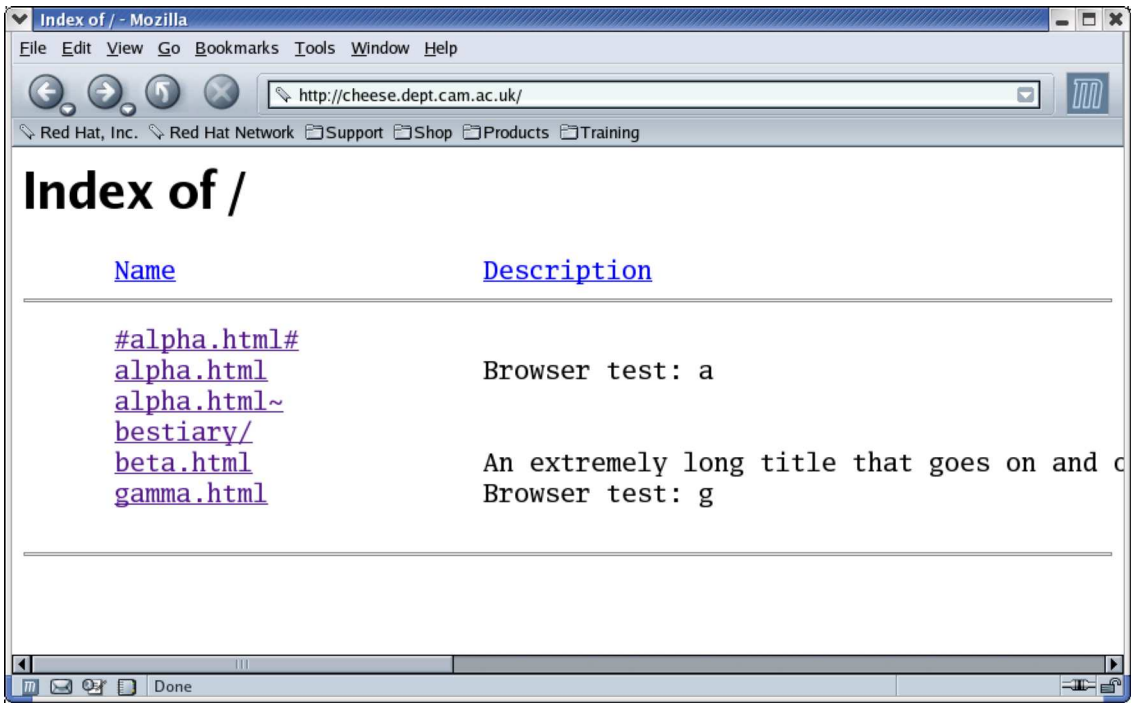

Figure 6-18. Unwanted files in the listing

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

Figure 6-19. `httpd.conf`: Ignoring certain files

Note that the two expressions to be ignored are placed in quotes. This is not typically necessary but under certain circumstances it is required, so the author tries to keep in the habit of doing it always. 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.

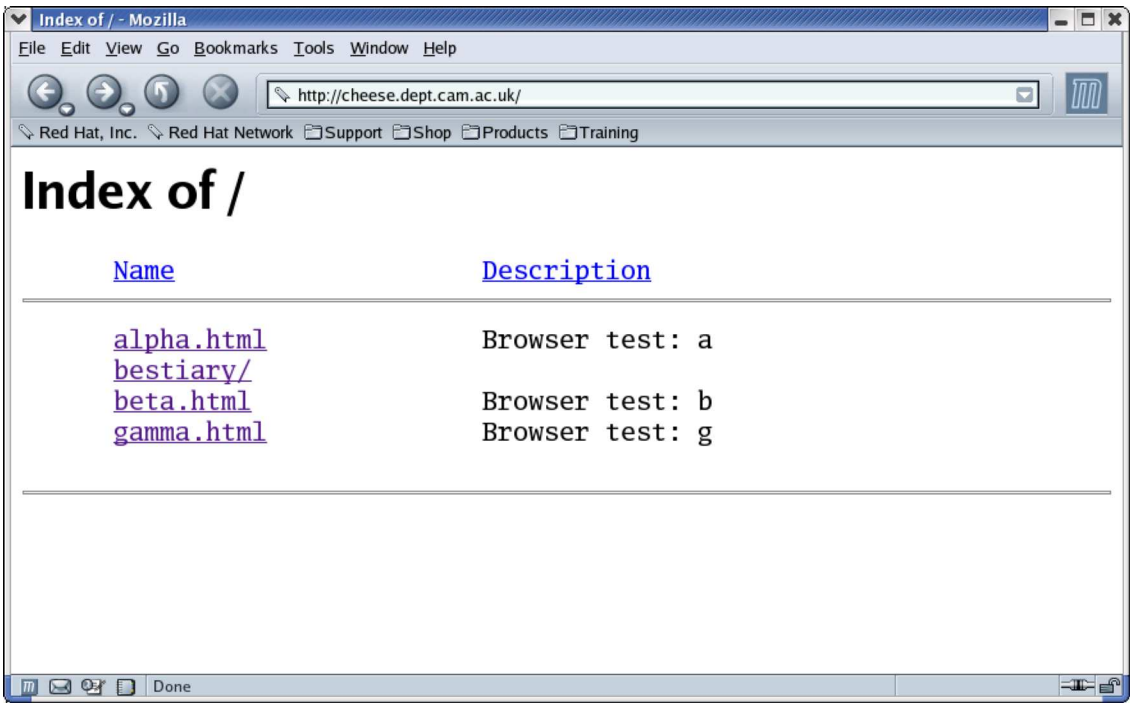

Figure 6-20. Unwanted files removed from the listing

> **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 based on a file's name* in Chapter 10.

We also have the issue of ordering to consider. At the moment the directories and plain files are intermingled. We can split these off with the **IndexOptions** `FoldersFirst` option. There is no equivalent `FoldersLast` option.

```
IndexOptions    FancyIndexing ... FoldersFirst
```

Figure 6-21. `httpd.conf`Putting subdirectories first in the listing

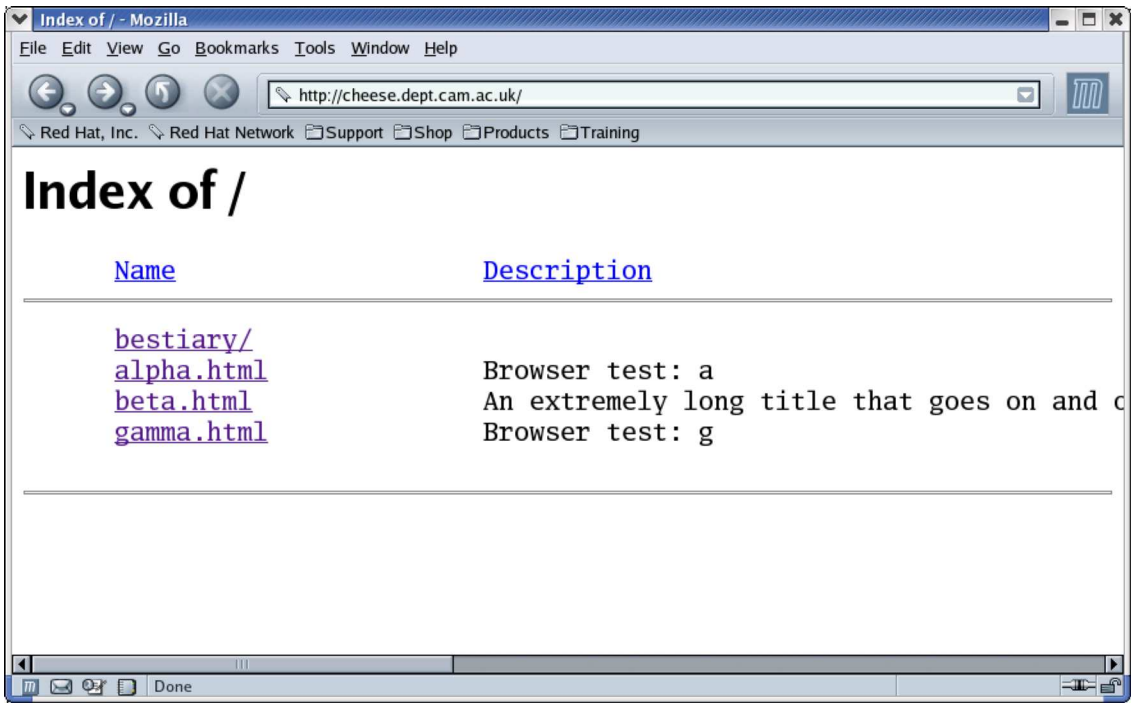

Figure 6-22. Putting subdirectories first in the listing

> **Note:** By now you may be starting to get confused about when a facility stands alone as a command like **IndexIgnore** and when it is an **IndexOptions** option like `FoldersFirst`. Me too.

Adding icons to the listing

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 `/var/www/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**. To make it available to all our web sites we are going to introduce another facility: aliasing. This comes courtesy of the alias_module module and its **Alias** command.

```
LoadModule alias_module modules/mod_alias.so
Alias  /icons/  /var/www/icons/
```

Figure 6-23. Setting up an alias

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 `/var/www/icons/`. Is 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.

```
AddIconByType   /icons/layout.gif       text/html
AddIconByType   /icons/text.gif         text/plain
AddIconByType   /icons/generic.gif      text/*

AddIconByType   /icons/image2.gif       image/*
AddIconByType   /icons/sound1.gif       audio/*
AddIconByType   /icons/movie.gif        video/*

AddIconByType   /icons/ps.gif           application/postscript
AddIconByType   /icons/pdf.gif          application/pdf

DefaultIcon     /icons/ball.gray.gif

AddAltByType    "HTML file"     text/html
AddAltByType    "Plain text"    text/plain
AddAltByType    "Text"          text/*

AddAltByType    "Static image"  image/*
AddAltByType    "Audio"         audio/*
AddAltByType    "Video"         video/*

AddAltByType    "PostScript"    application/postscript
AddAltByType    "PDF"           application/pdf
```

Figure 6-24. Specifying icons for MIME content types

> **Note:** Normally I would recommend using the PNG icons rather than the GIF icons to avoid possible future patent problems with Unisys. However, whoever did the conversion got the background transparency wrong and 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^^"
AddIcon         /icons/back.gif ".."

AddAlt          "Directory"     "^^DIRECTORY^^"
AddAlt          "Up"            ".."
```

Figure 6-25. `httpd.conf`: Support for directories

There is one last change we should make after all this playing with icons. An icon is an embedded image. It is generally regarded as a "good thing" to annotate the `<IMG/>` tag with the height and width of the image in pixels. We can specify these with the **IndexOptions** options `IconWidth` and `IconHeight`. These cause the web server to include `width="20" height="22"` in the `<IMG/>` tag's attributes. These numbers are wired into Apache and correspond to the icons it ships with, all of which are 20×22 pixels in size.

```
IndexOptions  FancyIndexing ... IconWidth IconHeight
```

Figure 6-26. `httpd.conf`: Specifying icon sizes

Conclusion

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

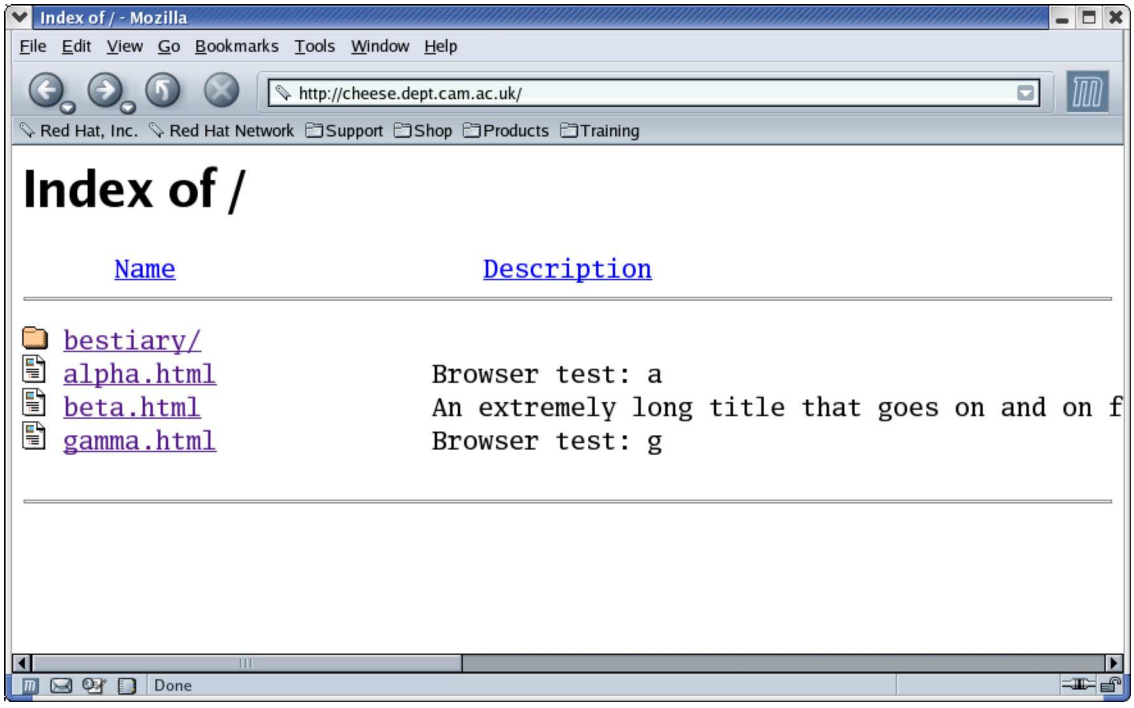

Figure 6-27. Listing with icons

Adding text to the listing

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
```

Figure 6-28. `httpd.conf`: Specifying a header file

```
<p>Here is an HTML fragment to appear above
the file listing.</p>
```

Figure 6-29. The `HEADER.html` file

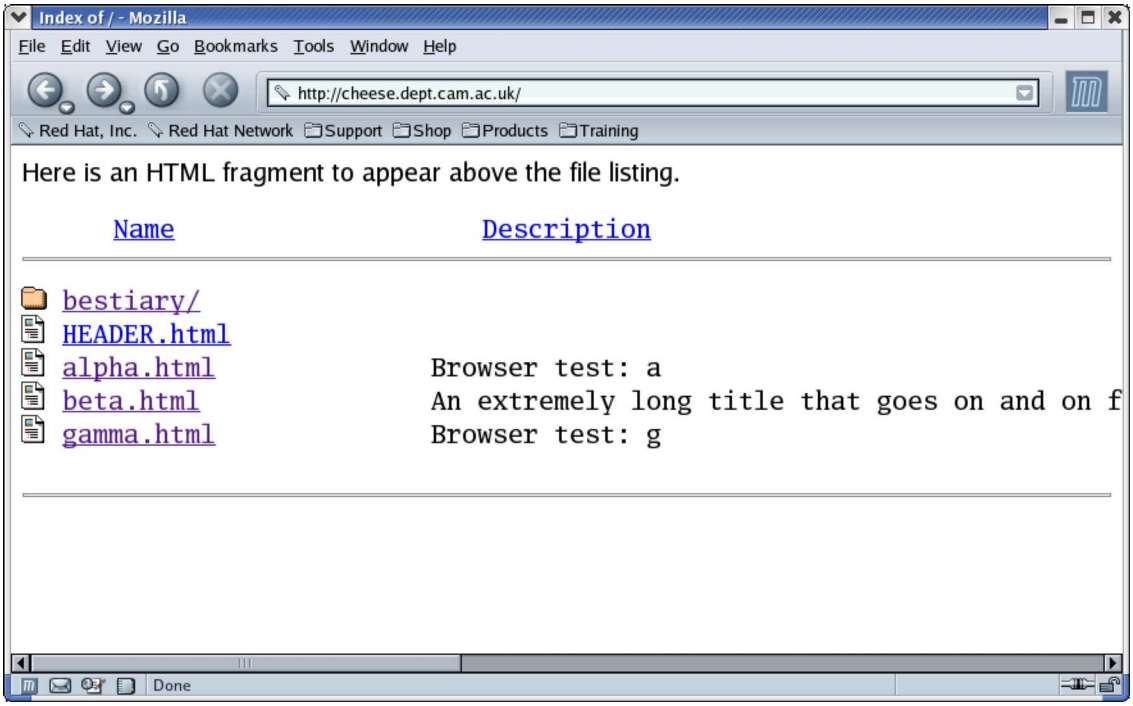

Figure 6-30. The file listing with header file included

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.

Sometimes, you *do* want to modify the HTML headers provided by the web server. For this purpose there is an option to the **IndexOptions** command: `SuppressHTMLPreamble`. With this option in place, the `HEADER.html` file should be a complete "top half" of an HTML document.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<title>An index page</title>
```

```
</head>
<body bgcolor="#ffffff" text="#000000"
 link="#003399" alink="#cc0000" vlink="#cc3333">
<p>Here is an HTML half page to appear above
the file listing.</p>
```

Figure 6-31. `HEADER.html` with headers

```
IndexOptions    FancyIndexing ... SuppressHTMLPreamble
HeaderName      HEADER.html
```

Figure 6-32. `httpd.conf`: Providing our own HTML header

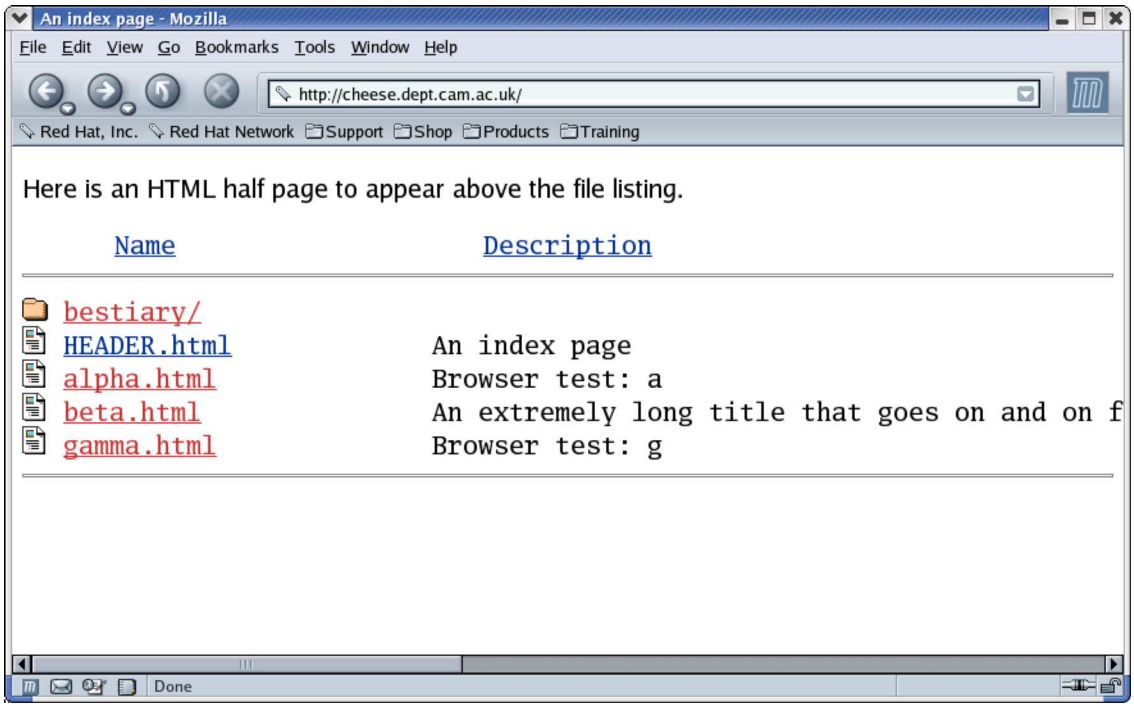

Figure 6-33. Links with different colours, courtesy of the `<BODY>` tag

In addition to placing text above the listing it is possible to place it underneath too. A sensible name for this would be a "footer" to correspond with "header". It's called a "readme". Ho hum. The corresponding command is **ReadmeName** and this is required to be an HTML fragment which does not contain the `</BODY>` or `</HTML>` tags.

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. (Red Hat Linux version 9 ships with 2.0.40). The author 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. If you use HTML tables (and the author thinks it is a good idea) then you still need to use the `NameWidth=*` and `DescriptionWidth=*` options to **IndexOptions**.

```
IndexOptions  ... HTMLTable ...
```

Figure 6-34. `httpd.conf`: Using HTML tables for the index

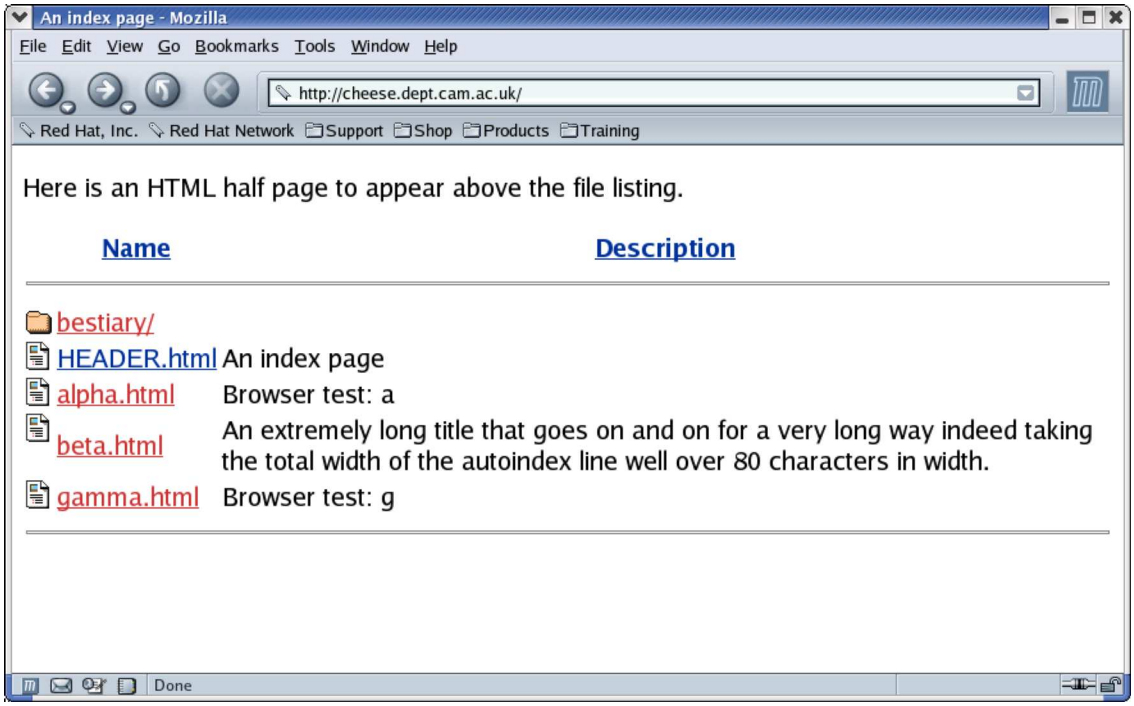

Figure 6-35. The listing in HTML table format

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 particlar 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 particualt 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.

`ScanHTMLTitles` (f)

The Description column is filled in with the HTML `<TITLE>`.

`SuppressSize` (f)

The Size column is not displayed.

`SuppressLastModified` (f)

The Last modified column is not displayed.

`SuppressDescription` (f)

The Description column is not displayed.

`FoldersFirst`

List the subdirectories before the plain files.

`SuppressHTMLPreamble`

Instructs the web server not to create the HTML preamble but to rely on the header file (identified by the **HeaderName** command) to provide it instead.

`NameWidth=`*`width`* (f)

Specifies the width of the Name column. If the width is a number then it is that many characters wide. If it is an asterisk then it is as wide as the longest name (with a lower bound).

`DescriptionWidth=`*`width`* (f)

Specifies the width of the Description column. If the width is a number then it is that many characters wide. If it is an asterisk then it is as wide as the longest description (with a lower bound).

`IconWidth` (f)

If icons are being used then this specifies that the web server should include the `WIDTH` attribute in the `<IMG/>` tag.

If it is used on its own then it inserts a width of 20 pixels. This is the width of every icon shipped in the `icons` directory. Alternatively, it can be used as `IconWidth=`*`N`* whereupon it will use *`N`* as the width. All icons must be of the same width; Apache has no mechanism to analyse icons on the fly.

`IconHeight` (f)

If icons are being used then this specifies that the web server should include the `HEIGHT` attribute in the `<IMG/>` tag.

If it is used on its own then it inserts a height of 22 pixels. This is the height of every icon shipped in the `icons` directory. Alternatively, it can be used as `IconHeight=`*`N`* whereupon it will use *`N`* as the height. All icons must be of the same height; Apache has no mechanism to analyse icons on the fly.

`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 "it depends on what order we load them in". If we load autoindex_module and then dir_module the latter is never used. The reason for this is that Apache runs though its modules *in order* looking for one that can handle the request. If a directory query is received then, even if the directory contains an `index.html` file, autoindex_module can handle it and does so.

If we load dir_module first then the situation is far more useful. If there is an `index.html` file then it is used. If there isn't then the directory is indexed automatically. But there's a little more. If both are in play then the `ScanHTMLTitles` option gives us a little more. If a subdirectory contains an `index.html` file then its `<TITLE>` is used as a description for the directory in the parent directory's automatic index.

```
LoadModule      dir_module      modules/mod_dir.so
DirectoryIndex  index.html

LoadModule      autoindex_module        modules/mod_autoindex.so
IndexOptions    FancyIndexing ScanHTMLTitles SuppressSize SuppressLastModified ...
```

Figure 6-36. `httpd.conf`: Running with both directory modules

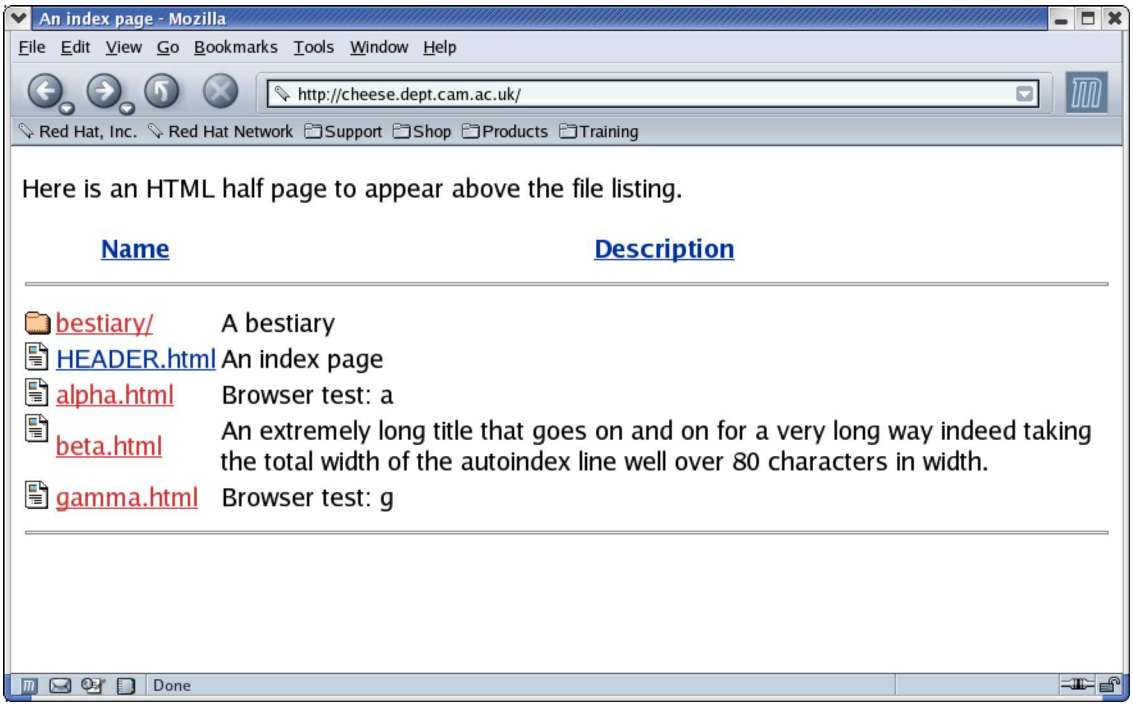

Figure 6-37. A listing with a directory's description

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 Red Hat Linux system-wide mechanism for log rotation and briefly mention, and then discard, an Apache-specific way to do the same thing.

Log file analysis

We will illustrate the use of the webalizer package for processing the log files to provide a set of web pages illustrating the use of the server.

Legalities

There will be a brief description of the legal implications of keeping log files. The author is not a lawyer and everything will change as either Parliament implements yet another piece of contradictory legislation or a senile fool of a high court judge farts while setting precedent.

The error log

The log file `/var/log/httpd/error_log` contains the error reports and warnings from the web server. By default, the directory `/var/log/httpd` 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.

```
[Mon Mar 24 10:23:11 2003] [notice] SIGHUP received.  Attempting to restart
[Mon Mar 24 10:23:11 2003] [notice] Apache/2.0.40 (Red Hat Linux) configured
 -- resuming normal operations
```

Figure 7-1. `error_log`: Reloads

Our first example will be seen in the log files from this course more than any other lines (we hope!). The line that starts "`SIGHUP received`" is the logged entry that means we requested a reload of the configuration file. A SIGHUP is an operating system signal sent to a running process (the web server) instructing it to do something. In our case it is to reread 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.

```
[Mon Mar 24 09:50:58 2003] [notice] caught SIGTERM, shutting down
```

Figure 7-2. `error_log`: Shutting down

Analogous to SIGHUP is SIGTERM which is another of these operating system signals, but one with a universal function. It is the instruction to a process to shut down.

```
[Wed Mar 26 09:25:50 2003] [error] [client 131.111.10.87] File does
not exist: /var/www/CHEESE/nonesuch.html
```

Figure 7-3. `error_log`: File not found

Another commonly occuring `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`*

Table 7-1. Logging levels

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/CHEESE/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 ...`"

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.

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 10 of /etc/httpd/conf/httpd.conf:
Invalid directory indexing option
```

Figure 7-4. `error_log`: Unformatted error messages

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

```
ErrorLog        /var/httpd/logs/error.log
```

Figure 7-5. `httpd.conf`: Logging to a different file

The **ErrorLog** directive gives the name of the error log file (relative to the server root, `/etc/httpd` if it not given as an absolute path). By default the log file is specified as `logs/error_log`. You will recall that `/etc/httpd/logs` is a symbolic link to `/var/log/httpd` so error logs are stored, by default, in `/var/log/httpd/error_log`. Any filename given to this command that does not start with a `/` will be resolved relative to the server root.

If the file name given is "`syslog`" then logging is not done to `/etc/httpd/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 (`/etc/httpd` by default).

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. Clearly 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"
```

Figure 7-6. `httpd.conf`: Basic logging of requests

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://cheese.dept.cam.ac.uk/`
- `http://chalk.dept.cam.ac.uk/`
- `http://cheese.dept.cam.ac.uk/gamma.html`

```
[08/May/2003:10:26:02 +0100] / 131.111.11.148 200
[08/May/2003:10:26:10 +0100] /index.html 131.111.11.148 200
[08/May/2003:10:26:29 +0100] /gamma.html 131.111.11.148 200
```

Figure 7-7. `request.log`: The corresponding output

Problems with the output as it stands

- There is no record of whether chalk.dept.cam.ac.uk or cheese.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.

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

```
<VirtualHost *>
ServerName      cheese.dept.cam.ac.uk
DocumentRoot    /var/www/CHEESE
CustomLog       logs/cheese.log "%t %U %h %s"
</VirtualHost>
```

Figure 7-8. Setting a log file for cheese.dept.cam.ac.uk

To enable the use of hostnames rather than addresses, we must instruct 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. It is also required if you plan to do any access controls based on host names as we will be in the Section called *Access control by 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
```

Figure 7-9. Turning on hostname lookups

```
[26/Mar/2003:14:09:54 +0000] / noether.csi.cam.ac.uk 200
[26/Mar/2003:14:09:54 +0000] /icons/dir.gif noether.csi.cam.ac.uk 304
[26/Mar/2003:14:09:55 +0000] /icons/layout.gif noether.csi.cam.ac.uk 304
[26/Mar/2003:14:09:57 +0000] /alpha.html noether.csi.cam.ac.uk 200
[26/Mar/2003:14:10:00 +0000] /beta.html noether.csi.cam.ac.uk 200
```

Figure 7-10. The cheese.dept.cam.ac.uk log file

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       logs/chalk_log          "%h %l %u %t \"%r\" %>s %b"
```

Figure 7-11. `httpd.conf`: Specifying the Common Log Format for chalk.dept.cam.ac.uk

```
noether.csi.cam.ac.uk - - [26/Mar/2003:14:25:49 +0000]
"GET /index.html HTTP/1.1" 304 0
noether.csi.cam.ac.uk - - [26/Mar/2003:14:26:00 +0000]
"GET /nonesuch.html HTTP/1.1" 404 211
```

Figure 7-12. The Common Log Format for chalk.dept.cam.ac.uk

The escape sequences used in the Common Log Format

%h

The client's hostname

%l

If the web server was doing IDENT (RFCnnnn) 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      chalk.dept.cam.ac.uk
DocumentRoot    /var/www/CHALK
CustomLog       logs/chalk_log    clf
</VirtualHost>

<VirtualHost *>
ServerName      cheese.dept.cam.ac.uk
DocumentRoot    /var/www/CHEESE
CustomLog       logs/cheese_log   clf
</VirtualHost>
```

Figure 7-13. Using named log formats with virtual hosts

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.

Finding bad links

Most browsers also include a query header Referer: which 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.

```
CustomLog       "%{Referer}i %U %>s"     referer_log
```

Figure 7-14. Logging referers

Those pages which have bad links pointing into your site will generate status code 404 and can be tracked down.

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, Red Hat Linux provides 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 Red Hat Linux'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 instructions tells it to rotate any file in `/var/log/httpd` whose name ends with `log`.

```
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

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

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

Figure 7-15. `/etc/logrotate.conf`: The main log rotation file

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.

`/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.

```
/var/log/httpd/*log {
    missingok
    notifempty
    sharedscripts
    postrotate
        /bin/kill -HUP ‘cat /var/run/httpd.pid 2>/dev/null‘ 2> /dev/null || true
    endscript
}
```

Figure 7-16. `/etc/logrotate.d/httpd`

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

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

/var/log/httpd/*log { ... }

This specifies that the commands within the curly brackets are to be applied to all the files that match the expression `/var/log/httpd/*log`.

missingok

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

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.

sharedscripts

Further down we will see a shell script that will be run after the rotatiopn has happened. This command instructs the system to run that script only once after all the files matching `/var/log/httpd/*log` have been rotated and not after each individual rotation.

postrotate ... endscript

Following the rotation of a log file (or all of them if **sharedscripts** is invoked) 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.

/bin/kill -HUP 'cat /var/run/httpd.pid 2>/dev/null' 2> /dev/null || true

This messy command sends a SIGHUP signal to the master web server demon. It is equivalent to **/etc/init.d/httpd reload** except that it sends the SIGHUP to *all* the web server processes.

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 *every* file in that directory. So if you edit the file `httpd` and leave behind both `httpd` and `httpd~` 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.

Log file analysis

Red Hat Linux ships with the webalyzer which is a log file analysis package that generates a graphical representation of the use of your web server. This can be particularly useful for convincing heads of department that web servers are actually used.

This course focuses on the use of the Apache web server so we will not look in depth at tuning the configuration to vary the format of the output. Instead we will simply focus on what it does and how to configure it to work with multiple virtual hosts.

The configuration file is `/etc/webalizer.conf` and is set up to match the Red Hat default file locations. As we have changed them we have to change this file too. If you are using the default Red Hat settings then skip this section and look in `http://server/usage/` for the usage statistics.

There are two approaches to changing this configuration for a multiple-site server. The first is to give each web site a copy of its own configuration file. This lets each site have full control over the presentation of the usage figures. The second approach is to have a common configuration file that lacks the per-site information and has the missing information provided on the command line. This lets each site rely on a centrally maintained, common presentation format.

In this course we will take the former approach for no better reason than that it's slightly easier to teach. This is a course about running `Apache`, not `Webalizer`.

`Webalizer` maintains a directory of pages and images (of usage graphs) to present the statistics. In the default Red Hat Linux installation this is maintained in `/var/www/html/usage`. We will, therefore, create `/var/www/CHALK/usage` and `/var/www/CHEESE/usage`.

```
# mkdir /var/www/CHALK/usage
# mkdir /var/www/CHEESE/usage
# chmod g+ws /var/www/CHALK/usage
# chmod g+ws /var/www/CHEESE/usage
```

Figure 7-17. Creating the usage directories

To keep things under the editorial control of the web site managers we will copy the `webalizer` configuration files into `/var/lib/webalizer` and change the group and permissions to give them control.

```
# cp /etc/webalizer.conf /var/lib/webalizer/chalk.conf
# cp /etc/webalizer.conf /var/lib/webalizer/cheese.conf
# chgrp chalk /var/lib/webalizer/chalk.conf
# chgrp cheese /var/lib/webalizer/cheese.conf
# chmod g+w /var/lib/webalizer/chalk.conf
# chmod g+w /var/lib/webalizer/cheese.conf
```

Figure 7-18. Copying the configuration files

Next, we are going to edit half a dozen items in each of the configuration files.

Parameters to change in the `webalizer` configuration file

LogFile /var/log/httpd/cheese_log

This specifies the log file to read the raw data from.

LogType clf

This identifies the format of the log file as common log format.

OutputDir /var/www/CHEESE/usage

This identifies the directory in which the statistics are to be compiled for presentation.

HistoryName /var/lib/webalizer/cheese.hist

This specifies a "working file" the system uses. By default it will appear in the output directory, but we don't want it in a downloadable location.

IncrementalName /var/lib/webalizer/cheese.current

This is another "working file" the system needs.

HostName cheese.dept.cam.ac.uk

This identifies the host name used to identify the site. For a virtual host it needs to be set.

Finally, all we need to do is to make sure that the log analysis program gets run over each of the web sites. It is run out of a daily cron job controlled by the file `/etc/cron.daily/00webalizer`.

```
#! /bin/bash
# update access statistics for the web site

if [ -s /var/log/httpd/access_log ] ; then
    /usr/bin/webalizer
fi

exit 0
```

Figure 7-19. The initial `/etc/cron.daily/00webalizer` file

We need to run it multiple times for our various web sites using its command line option to select non-standard configuration files.

```
#!/bin/bash

for conf in /var/lib/webalizer/*.conf
do
  /usr/bin/webalizer -c ${conf}
done

exit 0
```

Figure 7-20. Modified cron script

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 quesion, 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          modules/mod_userdir.so
UserDir         public_html
```

Figure 8-1. Loading and using the module

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.

Table 8-1. userdir_module lookups of `http://server/~bob/alpha/beta.html`

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.

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

The user directory example was the first where we were passing control outside our tidy document 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      cheese.dept.cam.ac.uk
  DocumentRoot    /var/www/CHEESE
  CustomLog logs/cheese_log clr

  <Directory /var/www/CHEESE/bestiary>
    DirectoryIndex main.html index.html
  </Directory>

</VirtualHost>
```

Figure 9-1. `httpd.conf`: Using the <Directory> command

The **<Directory *dir*>** ... **</Directory>** identifies a series of commands which should override or enhance the general settings for a specific subdirectory, `/var/www/CHEESE/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 cheese.dept.cam.ac.uk web site.

Figure 9-2. The `games` directory

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

```
<VirtualHost *>
  ServerName      cheese.dept.cam.ac.uk
  DocumentRoot    /var/www/CHEESE
  CustomLog logs/cheese_log clr

  <Directory /var/www/CHEESE/games>
    Options -Indexes
  </Directory>
 ...
</VirtualHost>
```

Figure 9-3. `httpd.conf`: Turning off indexes

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

Figure 9-4. Not the `games` directory

The **Options** command sets various parameters that basically control whether a module's or core facilities should be enabled or not. For the topic of delegation we will be interested in only three parameters. The first of these is `Indexes` which enables or disables autoindex_module. Note the leading minus sign. The **Options** command can be used in three ways, to disable a specific option, to enable a specific option or to set the complete set of options in one go.

Syntax summary: Setting options

Options -Indexes

Turn off automatic indexing. Leave all other options unchanged.

Options +Indexes

Turn on automatic indexing. Leave all other options unchanged.

Options Indexes

Turn on automatic indexing. Unset *all* other options.

So far, we haven't really delegated control. We have allowed for variation in subdirectories but we have not truly delegated the controls to people 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.

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

Figure 9-5. `httpd.conf`: Delegation to files in directories

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/CHEESE/games>** block to `/var/www/CHEESE/games/configuration` and the contents of the **<Directory /var/www/CHEESE/bestiary>** block to `/var/www/CHEESE/bestiary/configuration`.

```
IndexIgnore     "#*#" "*~" "configuration"
AccessFileName  configuration
<VirtualHost *>
  ServerName      cheese.dept.cam.ac.uk
  DocumentRoot    /var/www/CHEESE
  CustomLog       logs/cheese_log clr
</VirtualHost>
```

Figure 9-6. `httpd.conf`: Delegated control

```
Options -Indexes
```

Figure 9-7. `/var/www/CHEESE/games/configuration`

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

Figure 9-8. `/var/www/CHEESE/bestiary/configuration`

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

However, 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. This rather unsatisfactory command 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.

Mixed working

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

Blocked names

It is also possible to block a file from being downloaded at all based on its name.

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 of the client, browsing system is known. This address, or the hostname associated with it, 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 Cabridge 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.

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.

Access control by client IP address

As ever, this functionality is provided by a module: access_module from library `mod_access.so`.

```
LoadModule      access_module           modules/mod_access.so
```

Figure 10-1. Loading access_module

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

These restrictions need not cover the whole server, and typically don't. They can be placed within a **<Directory>** block or delegated configuration file to restrict access to only a subtree of the server. 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,Allow` statements. This is best illustrated with some examples.

```
<Directory /var/www/CHALK>
Order   Allow,Deny
Deny from csx.cam.ac.uk
Allow from cam.ac.uk
Deny from csi.cam.ac.uk
</Directory>
```

Figure 10-2. A `Allow,Deny` example

Table 10-1. The processing of a request by client gauss.csi.cam.ac.uk

Stage	Match?	State
Initial		All requests refused.
Allow from cam.ac.uk	Rule matches.	Access is allowed.
Deny from csx.cam.ac.uk	Rules does not match.	No change in state.
Deny from csi.cam.ac.uk	Rule matches.	Access is denied.
Final		Access is denied.

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.

If you want to pass control of this to the delegated configuration files then you must pass the `Limit` option to the **AllowOverride** command. This command must also appear in a **<Directory>** command.

```
LoadModule      access_module           modules/mod_access.so
<Directory /var/www/CHALK>
  AllowOverride   Limit
</Directory>
```

Figure 10-3. Allowing access control to be delegated.

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 and userid/password

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.

```
LoadModule      auth_module             modules/mod_auth.so
```

Figure 10-4. `httpd.conf`: Loading the user authentication module

To start with, we will need a module: 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* autmatically 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 chalk,cheese,webadmin rjd4
# mkdir /etc/httpd/access
# chgrp webadmin /etc/httpd/access
# chmod g+ws /etc/httpd/access
# ls -ld /etc/httpd/access
drwxrwsr-x    2 root     webadmin      4096 Apr 14 11:26 /etc/httpd/access
# touch /etc/httpd/access/passwd
# chmod g+w /etc/httpd/access/passwd
# ls -l /etc/httpd/access/passwd
-rw-rw-r--    1 root     webadmin         0 Apr 14 11:39 /etc/httpd/access/passwd
```

Figure 10-5. Creating a userid/password file and adding users

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.

```
$ htpasswd -m /etc/httpd/access/passwd rjd4
New password: password
Re-type new password: password
Adding password for user rjd4
$ cat /etc/httpd/access/passwd
rjd4:$apr1$kEDyP/..$n0DCjezTD.T.C.1s3td6..
```

Figure 10-6. Setting up users in the password file

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 disctionary 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 access_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.

```
<Directory /var/www/CHALK>
  AuthType      Basic
  AuthName      "Restricted area"
  AuthUserFile  /etc/httpd/access/passwd
  Require       valid-user
</Directory>
```

Figure 10-7. `httpd.conf`: Implementing the "valid user" policy

Syntax summary: implementing the "valid user" policy

<Directory /var/www/CHALK>...</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/httpd/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.

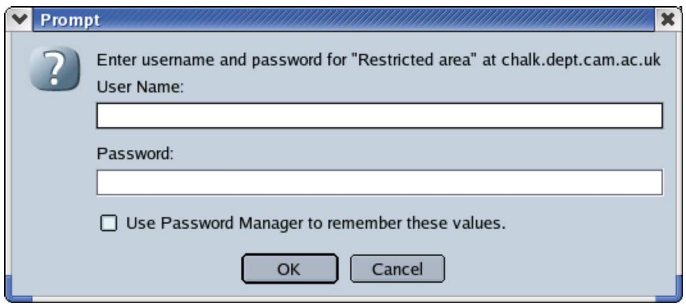

Figure 10-8. The userid and password challenge

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/httpd/access/passwd` file we can proceed.

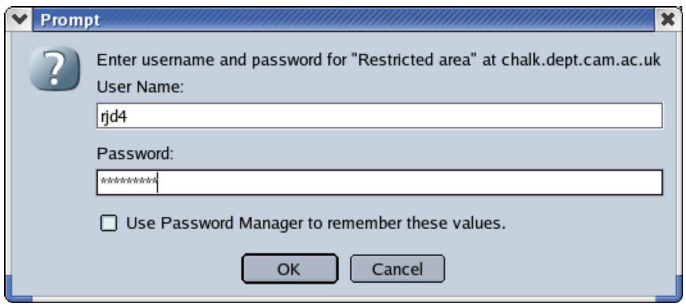

Figure 10-9. Entering the userid and password

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

```
<Directory /var/www/CHEESE/games>
  AuthType      Basic
  AuthName      "Cheese lovers only"
  AuthUserFile  /etc/httpd/access/passwd
  Require       user    tom  dick
</Directory>
```

Figure 10-10. `httpd.conf`: Restricting access to `/var/www/CHEESE/games` to users tom and dick.

The **Require user tom dick** 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.

```
stilton:        tom rjd4
cheddar:        tom dick
```

Figure 10-11. The `/etc/httpd/access/group` file

We can change from a user list to a group list by specifying which group file to use and which groups are permitted access.

```
<Directory /var/www/CHEESE/games>
  AuthType      Basic
  AuthName      "Cheese lovers only"
  AuthUserFile  /etc/httpd/access/passwd
  AuthGroupFile /etc/httpd/access/group
  Require       group   stilton  cheddar
</Directory>
```

Figure 10-12. `httpd.conf`: Restricting access to `/var/www/CHEESE/games` to groups cheddar and stilton

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 modules/mod_auth_digest.so

<Directory /var/www/CHEESE/games>
  AuthType      Digest
  AuthName      "Cheese lovers only"
  AuthDigestDomain      /games/
  AuthDigestFile        /etc/httpd/access/digest_pw
  AuthDigestGroupFile   /etc/httpd/access/group
  Require       group cheddar stilton
</Directory>
```

Figure 10-13. `httpd.conf`: The equivalent commands for the Digest protocol

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

```
$ touch /etc/httpd/access/digest_pw
$ htdigest /etc/httpd/access/digest_pw "Cheese lovers only" rjd4
Adding user rjd4 in realm Cheese lovers only
New password: password
Re-type new password: password
```

Figure 10-14. Adding a user to a digest password file

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 Basic authentication with passwords and groups held in other formats.

Table 10-2. Various storage formats for Basic authentication

auth_module	Text files. This is the module we have reviewed extensively above.
auth_anon_module	This acts in a manner analogous to anonymous FTP. It uses the userid anonymous and requests an email address for the password. This can be logged.
auth_dbm_module	This uses DBM files rather than text files. These are much faster to look up in once the number of ids stored becomes large.
auth_ldap_module	This uses LDAP to authenticate the user. This can be used to tie in with an existing LDAP service to stop your users needing to know yet another password. Unless you specify the use of LDAPS in the configuration files, the passwords will travel across the network in plain text from the web server to the LDAP server.
auth_mysql_module	This module checks the userids and groups against a MySQL database. This is not a standard Apache module but requires an additional software package.
auth_pgsql_module	As above, but using a PostgreSQL database.

Mix and match: Location and Authentication

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

Suppose we wanted http://cheese.dept.cam.ac.uk/games/ accessible from dept.cam.ac.uk without a password and with a password from elsewhere.

```
<Directory /var/www/CHEESE/games>
  Order Allow,Deny
  Allow from csi.cam.ac.uk

  AuthType      Digest
  AuthName      "Cheese lovers only"
  AuthDigestDomain      /games/
  AuthDigestFile        /etc/httpd/access/digest_pw
  AuthDigestGroupFile   /etc/httpd/access/group
  Require       group cheddar stilton
```

```
  Satisfy       any
</Directory>
```

Figure 10-15. Mixed restrictions

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 based on a file's name

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$)
```

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

Figure 10-16. Blocking access to the files ignored in the index

Chapter 11. Conclusion

What's next?

University Computing Service courses that have this course as a prerequisite.

Tidying up

Some re-ordering of the final configuration file.

We have illustrated a number of facilities in the Apache 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.

```
# These are the absolute basics to launch the web server.
Listen 80
User apache
Group apache
ServerRoot /etc/httpd

# Turn off all options for didactic reasons.
Options None

# Follow symbolic links
Options +FollowSymLinks

# Load the modules needed for this file
LoadModule mime_module modules/mod_mime.so
LoadModule dir_module  modules/mod_dir.so
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule alias_module modules/mod_alias.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule userdir_module  modules/mod_userdir.so
LoadModule      access_module           modules/mod_access.so
LoadModule auth_digest_module modules/mod_auth_digest.so

# Set up MIME type recognition by file name extension
TypesConfig /etc/mime.types

# Enable default documents for directory queries
DirectoryIndex index.html

# Enable automatic indexing of directories
Options +Indexes

# Make the indexes "fancy" and read HTML pages' titles for descriptions
# Don't show the size and timestamp columns.
# Let the name and description columns be as wide as they need to be
# Put folders first
# Use default sizes for icons
# Arrange that header files should have their own HTML preamble
IndexOptions FancyIndexing ScanHTMLTitles SuppressSize SuppressLastModified NameWidt
```

```
# Put a header file above the listing
HeaderName HEADER.html

# Set up aliasing
Alias  /icons/  /var/www/icons/

# Set up icons
AddIconByType /icons/layout.gif text/html
AddIconByType /icons/text.gif         text/plain
AddIconByType /icons/generic.gif      text/*

AddIconByType /icons/image2.gif       image/*
AddIconByType /icons/sound1.gif       audio/*
AddIconByType /icons/movie.gif        video/*

AddIconByType /icons/ps.gif           application/postscript
AddIconByType /icons/pdf.gif          application/pdf

DefaultIcon /icons/ball.gray.gif

AddAltByType "HTML file"     text/html
AddAltByType "Plain text"    text/plain
AddAltByType "Text"          text/*

AddAltByType "Static image"  image/*
AddAltByType "Audio"         audio/*
AddAltByType "Video"         video/*

AddAltByType "PostScript"    application/postscript
AddAltByType "PDF"           application/pdf

AddIcon  /icons/dir.gif "^^DIRECTORY^^"
AddIcon  /icons/back.gif ".."

AddAlt  "Directory" "^^DIRECTORY^^"
AddAlt  "Up"  ".."

# Suppress backup and working files from indexes
IndexIgnore "#*#"  "*~" "configuration"

# Delegate control via "configuration" files
AccessFileName "configuration"

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

# Set the error logging level to "info": informational messages.
LogLevel info

# Specify the error log file.
ErrorLog /var/log/httpd/error.log

# Set up access logging
# Split the log files between the two virtual hosts
# Define the Common Log Format for both of them
LogFormat "%h %l %u %t \"%r\" %>s %b" clf

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

# Users' own web pages
UserDir  public_html

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

```
NameVirtualHost *

<VirtualHost *>
ServerName chalk.dept.cam.ac.uk
DocumentRoot /var/www/CHALK
CustomLog logs/chalk.log clf
</VirtualHost>

<VirtualHost *>
ServerName cheese.dept.cam.ac.uk
DocumentRoot /var/www/CHEESE
CustomLog logs/cheese_log clf

<Directory /var/www/CHEESE/bestiary>
DirectoryIndex main.html index.html
</Directory>

<Directory /var/www/CHEESE/games>
Order Allow,Deny
Allow from csi.cam.ac.uk

AuthType      Digest
AuthName      "Cheese lovers only"
AuthDigestDomain      /games/
AuthDigestFile        /etc/httpd/access/digest_pw
AuthDigestGroupFile   /etc/httpd/access/group
Require       group cheddar stilton

Satisfy       any
</Directory>


</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 webservers, 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 vaious tasks, including dynamic page generation, processing fill-in forms, interfacing to databases, implimenting '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.

Appendix A. Apache modules

This lists the modules shipped with Red Hat Linux's packages.

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

Library	Module name	Description
`mod_access.so`	access_module	Access control by browser hostname.
`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.so`	auth_module	User authentication for access control usng plain text files..
`mod_auth_anon.so`	auth_anon_module	Allows anonymous user access and logs the password given.
`mod_auth_dbm.so`	auth_dbm_module	Just like auth_module but it uses DBM files rather than text files.
`mod_auth_digest.so`	auth_digest_module	Similar to auth_module but instead of using a plain text authentication scheme, it uses a cryptographic one.
`mod_autoindex.so`	autoindex_module	Automatically generates directory listings.
`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_dav.so`	dav_module	Distributed Authoring and Versioning functionality. (A standard for remote authoring and uploading.)
`mod_dav_fs.so`	dav_fs_module	A file system for DAV.
`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_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_headers.so`	headers_module	More general control of HTTP headers.
`mod_imap.so`	imap_module	Server-side image maps.
`mod_include.so`	include_module	Server-side includes.

Library	Module name	Description
mod_info.so	info_module	Lets the server report on its configuration via a web request.
mod_log_config.so	log_config_module	Configurable logging of requests and reponses.
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 conent negotiation between server and client.
mod_proxy.so	proxy_module	Lets your web server be a proxy. Typically it needs additional modules for specific protocols.
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_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 apache.
mod_unique_id.so	unique_id_module	Provides a unique key in the environemnt 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_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 Red Hat Linux packages that depend on the httpd package. Typically the package is named after the library. These are not sup[ported or maintained by the Apache group and there is no guarantee that they will not disappear between version of Red Hat Linux. The truly brave may care to wander through the Red Hat Linux "contributed" package sets for packages of Apache modules that aren't provided by Red Hat at all. *caveat administrator.*

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

Package	Library	Module name
mod_auth_mysql	`mod_auth_mysql.so`	mysql_auth_module
mod_auth_pgsql	`mod_auth_pgsql.so`	auth_pgsql_module
mod_perl	`mod_perl.so`	perl_module
mod_python	`mod_python.so`	python_module
mod_ssl	`mod_ssl.so`	ssl_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 outgoingt reponse 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*[1]) 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 paractice 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

Notes

1. http://www.w3.org/Protocols/rfc2616/rfc2616.html