Running a web server

***Apache* is the most popular web server in the world – why not try it yourself?**

ALTHOUGH EMAIL is a great way to avoid talking to people face to face, it was the world wide web that brought the internet to the masses. Early web browsers were capable of showing text and the odd picture, but since then we've seen the homepage revolution, the e-commerce revolution and now the blogging revolution. Each upheaval has had a huge impact on the way people use the web, and now, because you're running Linux, you can join the in-crowd of people serving up knowledge to the hungry public by running your own web server.

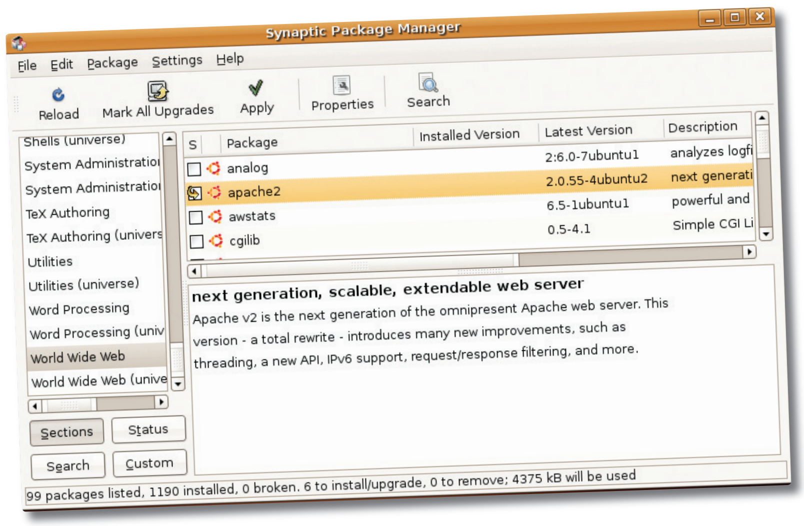
Although Windows dominates on the desktop, its dominance doesn't stretch to the server room. *Apache* – a web server that came about as a set of patches to the original NCSA web server – has gone from being A Patchy Web Server (from whence came its name) to being the number one web server in the world. At the time of writing, *Apache* powers nearly 70% of all web domains, compared with just 20% for Microsoft's *Internet Information Services*, its nearest competitor. And yet *Apache* also has a history of being secure and stable, which gives the lie to the "Windows gets hacked more because more people use it" argument.

In these pages you'll learn how to install and configure a web server on your local machine, how to secure it against the outside world and even how to install the PHP scripting language so you can create a dynamic website.

There are three main versions of *Apache*: 1.3.x, 2.0.x and 2.2. All are supported by the developers (when bugs are found, they'll fix them), all are patched if security vulnerabilities are found, but it's *Apache 2.2* that is recommended for use because it is the most modern release. There is little difference in performance between it and the others when running on Linux, so you can switch between them if you must.

INSTALLING APACHE

Apache is usually split into several packages in your package manager. The most important one is the



S	Package	Installed Version	Latest Version	Description
☐	analog		2:6.0-7ubuntu1	analyzes logfi
☑	apache2		2.0.55-4ubuntu2	next generati
☐	awstats		6.5-1ubuntu1	powerful and
☐	cgilib		0.5-4.1	Simple CGI Li

next generation, scalable, extendable web server
Apache v2 is the next generation of the omnipresent Apache web server. This version - a total rewrite - introduces many new improvements, such as threading, a new API, IPv6 support, request/response filtering, and more.

99 packages listed, 1190 installed, 0 broken. 6 to install/upgrade, 0 to remove; 4375 kB will be used

Use your package manager to search for *Apache* packages and install the ones you want. But avoid the temptation to pick everything on offer, as that will make your site less secure.

server itself, usually in a package called **apache** or **apache2**. Most distributions also usually include modules for *Apache* that add functionality, such as *mod_ssl* (for secure e-commerce), *mod_perl* (to use Perl scripts in your pages) and *mod_php* (to use PHP scripts).

Although it's tempting to install all of the modules on offer, you should be cautious. *Apache* is usually deployed as an outward-facing service and the more modules you deploy, the more open *Apache* is to attack by a malicious user. Install only what you use. For more advice, see the *Staying Secure* box on page 77.

Now you know what the packages do, go ahead and select *Apache* itself. Your package manager may prompt you to install other packages that Apache needs, so do that too. If you have it as an option, also select the **mod_php** package, as we'll be using that later.

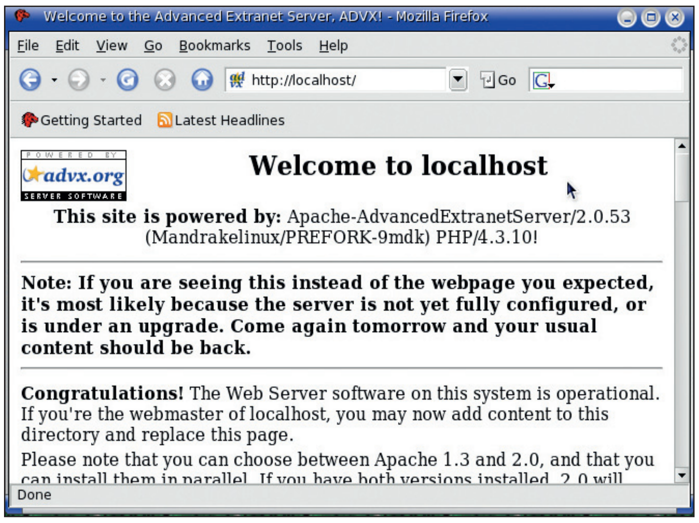
Once *Apache* is installed, you can test it out straight away. Open a terminal window and switch to root by typing **sudo /etc/init.d/apache2 start** to start *Apache* through its system startup script. It should default to a test page. This is accessible through your own IP address (on port 80 by default) or you can just use **localhost**. To try this out, open *Firefox* or any other web browser and enter **http://localhost**. All being well you should see something similar to the picture above right, which shows *Apache*'s test page.

"Although Microsoft dominates on the desktop, the open source Apache is the number one web server in the world."

The final test is to make sure that PHP has been installed correctly. To do this, start up your favourite text editor and create the file **test.php**. In there, type the following:

```
<?php
  phpinfo();
?>
```

The **phpinfo()** line is a special test function in PHP that outputs diagnostic information about your PHP installation. Save the file in your home directory and open a terminal. Use **su** to switch to the root account, then type **mv test.php /var/www/html**. That will move your new PHP script into the directory that *Apache* uses to serve its web pages. If the **/var/www/html** directory doesn't exist, you may need to look elsewhere – another popular location is **/var/www**.


Welcome to the Advanced Extranet Server, ADVX! - Mozilla Firefox

http://localhost/

Welcome to localhost

This site is powered by: Apache-AdvancedExtranetServer/2.0.53 (Mandrakelinux/PREFORK-9mdk) PHP/4.3.10!

Note: If you are seeing this instead of the webpage you expected, it's most likely because the server is not yet fully configured, or is under an upgrade. Come again tomorrow and your usual content should be back.

Congratulations! The Web Server software on this system is operational. If you're the webmaster of localhost, you may now add content to this directory and replace this page.

Please note that you can choose between Apache 1.3 and 2.0, and that you can install them in parallel. If you have both versions installed, 2.0 will

Done

Success! This test page shows that your *Apache* installation has worked, and that you're ready to run your own website.

With the file moved, open up a web browser and go to the URL **http://localhost/test.php**. If your install has worked, you should see a screen full of PHP information.

BASIC CONFIGURATION

Now that you have your *Apache* server up and running, you might want to make changes to the way it works. The basic configuration directory for *Apache* is usually either **/etc/httpd** or **/etc/apache** (depending on how your system is set up), but beyond that distros vary greatly. Usually you can rely on the main configuration file being called **httpd.conf** or **httpd2.conf**, but this may be in **/etc/httpd**, **/etc/httpd/conf**, or any number of similar paths. Start by looking in **/etc/httpd**, and if **httpd.conf** isn't in there, try some of the subdirectories.

The configuration file format is made up of sections and directives. A section starts with an opening angle bracket (<), the section name, and a closing angle bracket (>), then has a list of directives that apply only to that section. The section ends with another opening angle bracket, a slash (/), the section name again, and a closing angle bracket. If you've used HTML previously this format will be quite familiar.

Directives are the configuration options themselves. Any directive not inside a section is

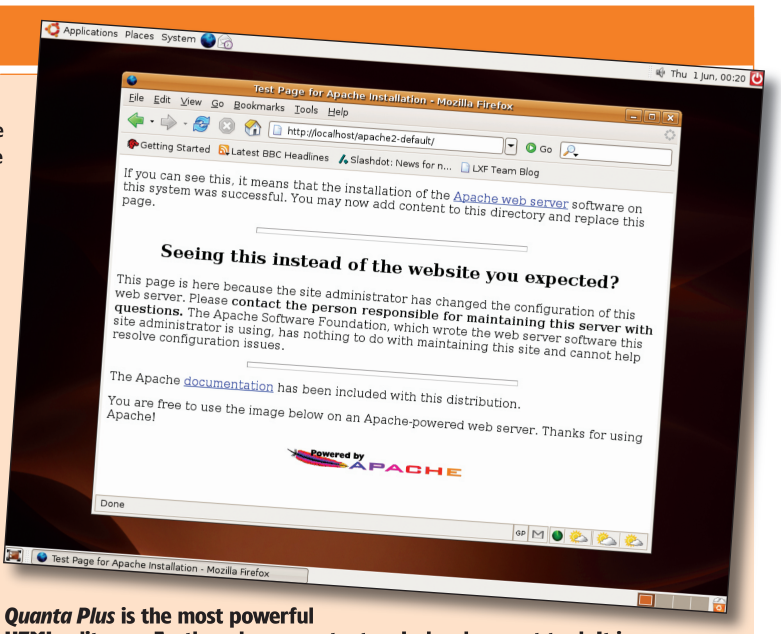
HOWTO... CREATE A WEBSITE

Although it might have seemed difficult, setting up your machine to serve web pages is actually the easiest part of running a website. The hard part is making the site good: creating an original, appropriate design, and adding content that people want to read. We can't help with the content – you need to write that yourself! – but we can at least point you in the direction of Linux's HTML editors so that you can create a site worthy of your words.

The most popular web editor on Linux is called *Quanta Plus*, which is part of the KDE desktop environment. Although it's very powerful, *Quanta* is a bit of an information overload – there are 13 different menu groups, four toolbars and four tab groups in its default display, which usually scares people off! However, when you get down to it *Quanta* is really little more than an advanced text editor that helps you write HTML for a website through a series of pre-made code snippets, buttons and wizards. Find out more about the aims and capabilities of the project at http://quanta.kdewebdev.org.

If you're not comfortable typing HTML, you can use *Quanta*'s Visual Page Layout view (although Quanta isn't the first thing we think of when we see VPL!) to drag and drop content on to your pages. This is a fairly new feature to *Quanta* and still needs some development (the undo function doesn't work, for example), but it's enough to get newcomers started.

If *Quanta* isn't your style, many distros come with alternatives, such as *Bluefish* and *Screem* – although these are inevitably a let-down after you've tried *Quanta*.


Test Page for Apache Installation - Mozilla Firefox

http://localhost/apache2-default/

If you can see this, it means that the installation of the Apache web server software on this system was successful. You may now add content to this directory and replace this page.

Seeing this instead of the website you expected?

This page is here because the site administrator has changed the configuration of this web server. Please **contact the person responsible for maintaining this server with questions.** The Apache Software Foundation, which wrote the web server software this site administrator is using, has nothing to do with maintaining this site and cannot help resolve configuration issues.

The Apache documentation has been included with this distribution.

You are free to use the image below on an Apache-powered web server. Thanks for using Apache!

Done

***Quanta Plus* is the most powerful HTML editor on Earth and a competent web development tool. It is pretty well established but its graphical features are still being perfected.**

applied to the whole server, whereas directives inside sections are only applied if the section is imported as a whole. For example:

```
PidFile /var/run/httpd.pid
DocumentRoot /var/www/html
<IfModule mod_rewrite.c>
   RewriteEngine On
   RewriteRule ^proxy:.* - [F]
</IfModule>
```

The first two lines are directives that aren't in any section, and so are applied to the whole of *Apache*. The first one, **PidFile**, dictates where *Apache* should store its process ID file – a file that a) tells the OS that *Apache* is running (and thus stops you from launching another *Apache* without any complex checking), and b) allows you to terminate *Apache* without knowing its process ID (you can just use the number stored in this file).

The second line is where *Apache* looks for its default set of HTML files. You should recognise the file path as being where we just stored our PHP script. **DocumentRoot** is a commonly changed directive, particularly when one server runs multiple sites, each with their own **DocumentRoot**.

Moving on, you can see there's a section immediately after **DocumentRoot** that starts with '**IfModule**'. Like many sections in *Apache*, this is a conditional section that only parses the directives inside if the module **mod_rewrite.c** is loaded. This is used because the next two directives only work if the *Rewrite* module is installed; without it they are meaningless.

THINGS TO CHANGE

There are five other popular directives that you might want to change before putting your site up. These are:

- **DirectoryIndex**.
- **HostnameLookups**.
- **KeepAlive**.
- **Listen**.
- **ServerAdmin**.

Some Linux distributions may place these directives into files other than **httpd2.conf**. If you have a **commonhttpd.conf** file you should try looking in there too.

The **DirectoryIndex** directive lists the files that should be served if none is specified in the URL. For example, if a visitor to your site requests **www.yoursite.com**, does *Apache* serve **index.html**, **index.php** or something else? To use **DirectoryIndex**, type the list of filenames that you consider default pages, separated by spaces:

```
DirectoryIndex        index.html index.htm
index.php default.html default.htm default.php
```

Using this line, when *Apache* hasn't been asked for a specific page to serve it will try to serve **index.html**. If the file doesn't exist, it will try **index.htm**, then **index.php**, **default.html** and so on until one of the files does exist, and that's what it will serve. This means that using **default.php** for the default page in directories is very inefficient: *Apache* will try five other filenames first, wasting a lot of CPU time. If none of the files is found, *Apache* will just list the contents of the directory.

The **HostnameLookups** directive is usually disabled by default, because no medium-sized website can afford the performance hit it causes. When a visitor comes to your site, *Apache* stores their IP address, among other things. Enabling **HostnameLookups** allows *Apache* to look at the IP addresses as they come in and figure out what domain they come from.

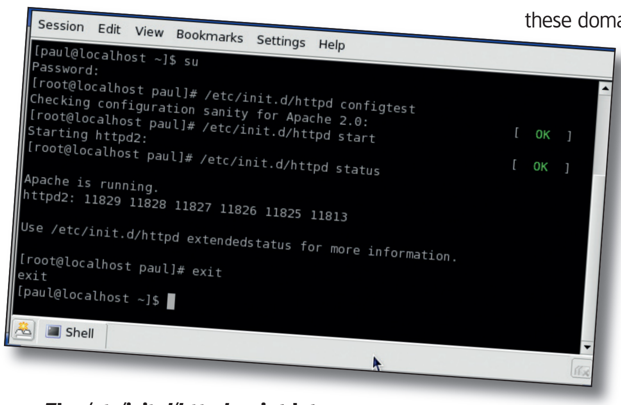
"HostnameLookups will look at the IP address of a site visitor and figure out what domain they came from."

So, rather than seeing that someone at 212.119.50.36 came to your site, it might say that someone from **microsoft.ca** visited – you get the name of the company, plus the country they visited from. This is great for analysing the profile of your website's visitors, but the extra work of resolving all these domains is time-intensive, so many people try to do it later on, during a quiet period. That said, resolving domains later on has a lower success rate because your visitor is not likely to be online so the domain lookup will fail for their IP.

The third directive, **KeepAlive**, can both improve and hurt performance. When a visitor requests a page from your site, a new connection is created for it. *Apache* receives the request, sends the page and closes the connection. On the client side, the HTML gets read and interpreted. Let's say the client notices it needs five pictures and a CSS stylesheet in order to display the page properly. It would open six more connections to receive each of these.

Opening and closing internet connections is a slow process, so **KeepAlive** was born. When a connection is opened for a web page, that page is transferred as normal. However, when the page has been sent, *Apache* gives the user 15 seconds to request other content over the connection. So, having received the HTML page and discovered it needs extra images, the client can request each of those images over the same connection and save all the extra opening and closing.

This is a huge speed boost for small- to medium-sized sites, but it becomes a problem for larger sites. *Apache* has a hard limit on the number of connections it can use at a time – usually between



```
[paul@localhost ~]$ su
Password:
[root@localhost paul]# /etc/init.d/httpd configtest
Checking configuration sanity for Apache 2.0:
[root@localhost paul]# /etc/init.d/httpd start                    [  OK  ]
Starting httpd2:
[root@localhost paul]# /etc/init.d/httpd status                   [  OK  ]

Apache is running.
httpd2: 11829 11828 11827 11826 11825 11813

Use /etc/init.d/httpd extendedstatus for more information.

[root@localhost paul]# exit
exit
[paul@localhost ~]$
```

The */etc/init.d/httpd* script lets you check your configuration, read server status and also start and stop *Apache*.

HOWTO... ADD USERS

Once you have your main website online, you're only a small step away from allowing others to have their own pages on your site. You need not grant them access to your precious site, because *Apache* allows them to serve up pages directly from their home directory.

To try this out for yourself, go to your home directory and create the directory public_html. In there, create the file hello.php and type this in:

```
<?php
   echo "Hello, world!";
?>
```

Save the file, then open your web browser. Enter the following URL into the location bar: http://localhost/~paul/hello.php. You will of course need to replace 'paul' with your username, but the tilde beforehand tells *Apache* that this is a username and that it should look in that user's public_html directory for the requested file.

If the request fails, you may need to grant users access to public_html. To do that, open up a terminal and type chmod o+r public_html. The o stands for 'others', and the +r means 'allow reading'.

150 and 200. When using **KeepAlive**, the 15-second timer is reset each time a new request is made over the connection, which means that once the final image has been sent to the client there's a 15-second gap where nothing is done before the connection is closed. If you have thousands of visitors coming to your site, holding connections open for 15 seconds is a huge waste. In these circumstances you should either cut the **KeepAlive** timeout or just disable **KeepAlive** altogether.

The **Listen** directive is where you specify the port number on which *Apache* should listen for requests. By default this is port 80, which means that any requests that come through on that port will be routed to *Apache* for handling. Choosing a port number below 1024 requires root-level privileges, so many people who run a web server on machines where root access isn't available choose port 8000 or 8080.

If you want to change the port number, be advised that although you *can* choose a port below 1024 if you have root access, you should be careful that it doesn't clash with other services. FTP usually runs on port 21, SSH on port 22, SMTP on port 25, IMAP on port 143 and so on – there are many services assigned by default to use ports below 1024, and trying to change *Apache* to another sub-1024 port may cause a clash. Instead, we'd advise you to pick a port over 1024; you can usually choose these with impunity.

Finally, the **ServerAdmin** directive lets you define an email address that *Apache* will use for error pages as a contact address. If you want to receive emails from users who find error pages, you should set this to something valid. If you would rather not hear from users – and who does? – set this to an email address that goes nowhere.

Although there are many other configuration options that we haven't covered here, you can get by without most. If you want to get into more advanced web serving – particularly if you want to host more than one domain – you should be prepared to read the lengthy *Apache* documentation, or invest some money in a good *Apache* book. ●

STAYING SECURE

As an internet-facing service, *Apache* is vulnerable to attack from malicious visitors. Although it is impossible to have a fully secure system, there are a number of measures you can take to minimise your risk. Here are our top seven recommendations for keeping secure with *Apache*.

1 Install only what you need. There are many modules available for *Apache*, but each one that you add increases the likelihood of someone successfully attacking you. If you just want to serve plain HTML pages, for example, why bother having PHP installed?

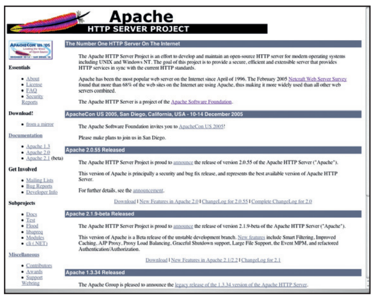
2 Update *Apache* regularly. Although security is really something you should closely monitor if you value your site, the most important rule is to ensure that *Apache* is kept up to date. It is generally considered a secure server, but new *Apache* security vulnerabilities are discovered frequently. Fortunately, patches to solve these problems are issued almost immediately by the developers and are usually made available inside the package manager, and you should update to them immediately. Even if you don't think you are directly affected, you may have a user on your site who is – and that's usually all it takes.

3 Restrict sensitive server information. By default, *Apache* sends out its name, version number and module information along with every request. This is great for companies such as Netcraft that want to gather and analyse this information, but it's also great for hackers that are scanning for vulnerable *Apache* versions. You can stop *Apache* from sending out all this information by changing the **ServerTokens** directive to Prod rather than Full, and **ServerSignature** to Off.

4 Be careful of other users. If you want to give other users access to your *Apache* server for their own files, be aware that scripting languages such as PHP and Perl are capable of taking up an inordinate amount of resources, accessing local commands, and otherwise causing havoc if your users are abusive.

5 Read your log files carefully. *Apache* keeps logs of all requests in **access_log**, and errors in **error_log**. Both of these files are usually found in **/var/log/httpd**. The **access_log** file is great for letting you know which pages are popular, but you can also use it to spot people requesting unusual pages or sending malformed requests – a sure sign they are up to no good. The **error_log** file is more helpful for spotting a bad configuration on your system, but occasionally you'll find it listing a page request for a page that doesn't exist. This might be pure chance, or it might be someone trying to exploit your server – look into it.

6 Keep track of announcements. Security holes, updates, fixes and workarounds are usually made public on the main *Apache* site, **http://httpd.apache.org**, but you can also check **www.apacheweek.com** and **www.serverwatch.com** to read technical articles and news about *Apache*. If you don't, you can be assured that hackers do.



Apache
HTTP SERVER PROJECT

Essentials
- About
- License
- FAQ
- Security Reports

Download!
- from a mirror

Documentation
- Apache 1.3
- Apache 2.0
- Apache 2.1 (beta)

Get Involved
- Mailing Lists
- Bug Reports
- Developer Info

Subprojects
- Docs
- Test
- Flood
- libapreq
- Modules
- cli (.NET)

Miscellaneous
- Contributors
- Awards
- Support
Webring

The Number One HTTP Server On The Internet

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

Apache has been the most popular web server on the Internet since April of 1996. The February 2005 Netcraft Web Server Survey found that more than 68% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined.

The Apache HTTP Server is a project of the Apache Software Foundation.

ApacheCon US 2005, San Diego, California, USA - 10-14 December 2005

The Apache Software Foundation invites you to ApacheCon US 2005!

Please make plans to join us in San Diego.

Apache 2.0.55 Released

The Apache HTTP Server Project is proud to announce the release of version 2.0.55 of the Apache HTTP Server ("Apache").

This version of Apache is principally a security and bug fix release, and represents the best available version of Apache HTTP Server.

For further details, see the announcement.

Download | New Features in Apache 2.0 | ChangeLog for 2.0.55 | Complete ChangeLog for 2.0

Apache 2.1.9-beta Released

The Apache HTTP Server Project is proud to announce the release of version 2.1.9-beta of the Apache HTTP Server ("Apache").

This version of Apache is a Beta release of the unstable development branch. New features include Smart Filtering, Improved Caching, AJP Proxy, Proxy Load Balancing, Graceful Shutdown support, Large File Support, the Event MPM, and refactored Authentication/Authorization.

Download | New Features in Apache 2.1/2.2 | ChangeLog for 2.1

Apache 1.3.34 Released

The Apache Group is pleased to announce the legacy release of the 1.3.34 version of the Apache HTTP Server.

The main *Apache* website.

7 Have a recovery plan ready. If your server is compromised – either through a hole in one of your pages or through *Apache* itself – you need to be ready to respond. Can you afford to simply shut the server down and start from scratch? Do you have backups of key files? Will you be able to tell if any important files have been copied or modified? Sometimes leaving the server running while it is compromised is the best move, as it allows you to watch the hacker return (which they invariably do) so that you can see where they connect from and what they are doing on your system.

HOWTO... GET IN DEEPER

If you're interested in seeing the communication between web client and web server, you can use the Telnet tool to simulate a HTTP request and response. Not all distros have this installed by default, so you should check your package manager for the *telnet* package before continuing.

To send an HTTP request to your server, open up a terminal, type telnet localhost 80, and press Enter. This will connect to the local computer on port 80, which is the default *Apache* port.

Next, type GET / HTTP/1.0 and press Enter twice. This requests the default page for your server, and pressing Enter twice signals that your request is complete and you're waiting for the page. *Apache* should respond almost immediately with the HTML for that page, but if you scroll upwards in the output you should see the HTTP headers that *Apache* sent back. These usually include the size of the page, the name of the server, the time and date, and more.