First pitch at Apache

Nigel Whitfield adds some strings to your bow, with his guide to configuring a web server.

So, you think you want a web server? Whether you want to make information available to colleagues on an office network, are trying to get your company online as an ebusiness, or want to test web development ideas without uploading them over a modem, a web server can be useful.

While there are web servers – notably Microsoft's Personal Web Server and Internet Information Server – that will run under Windows, the bulk of the web uses other systems, Apache web server for Unix or Linux being the clear leader.

Choosing Apache will therefore give you few compatibility problems and a wealth of expertise to tap into on the web. It's also free, and compared to the bloated, graphically-based alternatives, you can run it on a fairly low-powered system, and still have acceptable results. You can even run it on a Windows system which, while not ideal, will let you test the more obscure configuration options, or set up a company intranet.

The step-by-step guide opposite shows how to get a basic configuration of Apache running, and Tim Anderson's *Hands On Web Development* this month explains how to use the authorisation features in Apache to create restricted areas on your website. There's much, much more that you can do with it, though, and we'll explain some of the basics here. Apache is well documented too, and the default configuration is – unlike some other packages from the Internet – fairly safe.

One of the reasons that Apache is the most popular server on the web is that it's also one of the most flexible – you can use it as a proxy server, to host multiple websites on a single system with virtual servers, and extend it with loadable modules, such as the Perl module that allows you to control most of the server behaviour through your own scripts.

While most of what's in this article will work on any Apache installation, it has been written with Linux/Unix users in mind, since that's the best way to run a web server such as this; if you don't already have a machine set up, now's the time to bite the bullet and put a system on your network to run Apache. If you don't already have a TCP/IP network running between your systems, you're going to need it now.

One of the most common tasks when configuring a server is setting up virtual servers – which allows the same web server to host two or more completely separate sites. You might decide, for example, to have a website visible to the public at www.yourcompany.com, and an intranet, with office memos and other information at info.yourcompany.com. With Apache, both can be set up using the same server, on the same machine.

First, though, you'll need to decide how you want to manage the servers – you can have virtual servers, using an individual Internet (IP) address for each one, or differentiate between them by name alone. While the former requires an additional address, the latter only works with web browsers using HTTP 1.1. That includes most modern browsers – but some people may still have problems accessing your site via proxy servers.

One reason Apache is the most popular web server is that it's also one of the most flexible

To add a virtual server with its own IP address, you'll need to make sure that the system running Apache is listening to that address on the network as well, and in the httpd.conf file, tell the server either to listen to all addresses, or specifically add the ones you want, using Listen directives, such as:

```
Listen 192.168.1.1
Listen 192.168.1.7
```

You can then simply add a virtual server in either httpd.conf or srm.conf by defining it like this:

```
<VirtualServer 192.168.1.7>
ServerName ↙
www.yourcompany.com
DocumentRoot /usr/↙
local/webdocs/yourcompany
ServerAdmin ↙
webmaster@yourcompany.com
</VirtualServer>
```

(*Key: ↙ code string continues*)

Although it's possible to give hostnames in the VirtualServer directive, rather than an IP number, the latter is better for technical reasons. You can use most of Apache's configuration options within the VirtualServer section, allowing you to specify aliases, security and the format and location of log files.

To have multiple servers using the same address, the configuration is almost identical. You'll need, however, to tell Apache the IP address that you're using, with a line like this in httpd.conf:

```
NameVirtualServer ↙
192.161.1.1
```

For each VirtualHost directive that you create with the same IP address, you can specify a different location for the web documents, and all the other options you might want.

One common configuration, for example, is to create your own pages for error responses – especially for the 404 error that happens when someone mis-types a URL.

You'll find the options for this sort of thing in the srm.conf file, stored in the Apache configuration directory. If you've designed a custom page, perhaps with hot links to key parts of your site, and called it sorry.html, this is how you'd tell the server to use it, instead of the default – and remember, you can have a different page for each server on your system:

```
ErrorDocument 404 ↙
/sorry.html
```

You could even write a script that tries to guess what people were looking for, and make Apache run that, in just the same way.

Running scripts – which is probably the major reason you're considering running your own web server – can be done in a number of ways. You can gather all the scripts together in a single

Step-by-step guide to setting up an Apache web server

1 The Apache web server is downloadable from www.apache.org. You should download the latest version of Apache 1.3. You can compile it yourself, but to save time there are binary versions available for lots of operating systems, including Linux and Solaris. You'll also find it on other sites, such as Sun's Solaris Freeware archive. Download the file, and uncompress it, using gunzip.

2 Now you'll have to install. Assuming you have a file, called apache-1.3, the exact command depends on your computer system. Here, we're installing on Solaris, using the pkgadd command:

```
pkgadd -d apache-1.3
```

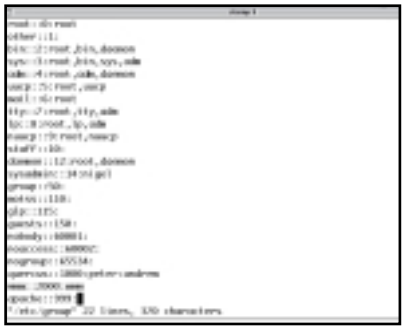
On a RedHat system, you'd type:

```
rpm --install apache-1.3
```

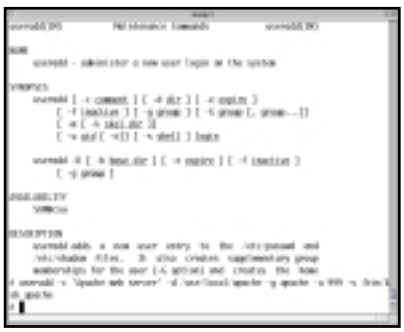
and on a SCO Unix system:

```
scoadmin software
```

starts the menu driven installer.



3 To run a web server safely on your computer, it should be configured with a user name and group ID that are unique – although you could use the 'nobody' or 'nouser' id if it already exists on your system. To create a new user, first pick a user and group ID that's free. Add the group ID to the /etc/group file; here we've added a group called apache, with the ID 999.



4 Now you can add the user; the exact syntax of the command will depend on your system, but the command is usually useradd. Set the home directory of the user and ID to be the main directory that the installer puts the web server in. We'll assume that's /usr/local/apache. You can set a login shell for the user, but it would be best to set it to something like /bin/false, for security.

5 Before you can start the server, you need to edit the configuration files. These will be in the etc directory, for example /usr/local/apache/etc. If there are no .conf files, rename the .conf.default ones. To begin with, you need to check a few basic settings, in httpd.conf, starting with Port. The default setting in the file is 8080. To serve pages without needing a port number in URLs, change it to port 80.

6 Now, find the User and Group entries and make sure that they match the ID you've decided to run the server as. Remember that in the event of a security breach, this user could have access to files on the system, so never run the server as a privileged user. If you start the server as root – essential if it's running on port 80 – it will change to this user when it's running.

(Turn over for steps 7-12)

place, which makes it much easier to manage them – and they'll be available to anyone who's using the server. The step-by-step guide shows how to enable these global cgis, via an alias to /cgi-bin/. If you're security conscious, you might want to change that to something else. Some 'attack scripts' look for programs in your /cgi-bin/ directory; so, change its name, for instance, to /web-scripts/ and running a standard script against your server will no longer be enough to exploit the problem. Thankfully, the latest versions of Apache don't have such problems – but if you're installing over an older one, you should make sure you know what all the programs in your script directory are for!

If you want to allow scripts elsewhere, perhaps for files with the extension .cgi, then you need to edit a few other things. Firstly, in the srm.conf file, use the AddHandler directive to tell Apache to use the CGI interface:

```
AddHandler cgi-script .cgi
```

You'll also need to set appropriate permissions in the access.conf file, either

Setting up Apache (continued)

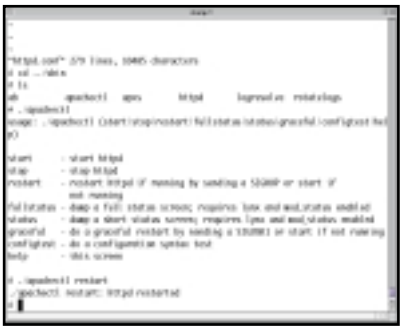
7 The ServerAdmin line in the file lets you give an email address; and this will appear automatically in some results pages, so enter something like webmaster@yourdomain. Check that the ServerRoot entry that follows is correct for your system – it should be the top level installation directory, such as /usr/local/apache. If it's wrong, the server won't be able to find most of the support files it needs.

8 Further down the file, you can set the server name – this allows the server to respond when it's called, for example, www.vnunet.com, even if it's running on a machine called fred. For the time being, just check that the intermediate entries for the location of log files are OK. At the moment, you don't really need to worry about checking the other information – the default will usually be fine.

9 Now, change to the sbin directory, usually /usr/local/apache/sbin. The apachectl program lets you start and stop the server. Type ./apachectl configtest to check that there are no serious errors in the configuration files, and correct any if there are. If the configuration is OK, then you can start the server by typing ./apachectl start.

10 Now fire up a web browser, and point it at the system you've installed Apache on. If all is well, you should see a page like this one, confirming that the system is running the web server. All the documentation is online, so you should read through it and familiarise yourself before configuring more options.

11 Now you know that the server is running, you can configure other options. In the srm.conf file, the DocumentRoot directive lets you set which directory is used for your web pages. You may want to change it from the default. And if you're creating files on PCs, edit the DirectoryIndex line, to add index.htm as well as index.html to the list of files used when a url ends in a /.



12 To enable Unix scripts on your server, find the ScriptAlias line in srm.conf, and edit it to point to a directory where your scripts will be kept – but only keep necessary files in there; every script is a potential security hole. When you've made the configuration changes that you want, use the command apachectl restart to make the server reload its configuration. Remember, if you changed the DocumentRoot, you might not have access to the documentation any more.

globally or, more securely, for directories that you want to allow scripts in:

```
<Directory /usr/local/webdocs/nigel/scripts>
Options ExecCGI
</Directory>
```

This example will allow files to be executed, if they're in that directory – the exact URL will depend where the directory is, relative to the DocumentRoot you set when you configured the server. For example, if DocumentRoot is /usr/local/webdocs, then the scripts would be /nigel/scripts/scriptname.cgi. Any .cgi files elsewhere simply wouldn't be allowed to run.

Apache is a huge and complicated program. There are many configuration options, and we've only scratched the surface here. The rest is up to you.

CONTACTS

Nigel Whitfield welcomes your comments. Contact him via the *PCW* editorial office or email **internet@pcw.co.uk**