Elective Courses

Hamish Whittal
Matthew West

Elective Courses

by Hamish Whittal and Matthew West

Published 2005-01-25 21:25:17

Table of Contents

List of Figures

This page intentionally left blank

Chapter 1. Apache

Introduction

Apache is a web server, that has it's roots in the CERN web server, but naturally has come a long way since those early days. It is the most widely used web server on the Internet today, and it is unlikely that much will topple it from that position in the near future. It's widespread use can, in some part, be attributed to the ease with which it can be integrated with content technologies like Zope, databases like MySQL and PostgreSQL and others (including Oracle and DB2) and the speed and versatility offered by Web rapid application development (RAD) languages like Personal Home Page (PHP). It is highly configurable, flexible and most importantly, it is open. This had lead to a host of development support on and around Apache. External modules such as mod_rewrite, mod_perl and mod_php have added fist-fulls of functionality as well as improved the speed with which these requests can be serviced. It has, in no small part, played a role in the acceptance of the Linux platform in corporate organizations.

In this module, we will learn how to configure and optimize Apache to serve web pages - it's primary function (although by no means it's only use). We will also learn how to write some simple CGI scripts for use in generating interactive web pages. But enough. Let's dive right in.

The Basics

The protocols

The world wide web runs on a protocol known as http or the hypertext transport protocol. This was first developed by Tim Berners-Lee in 1989. His initial idea was to provide a mechanism of linking similar information from disparate sources. Similar in concept to the way your mind might work. Http is the protocol that is the most utilized on the Internet today. It is still part of the TCP/IP protocol stack, just the same as SMTP, ICMP and others, but's it's responsibility is transferring data (web pages) around the Internet.

Apache versions

Apache comes in two basic flavors: Apache version 1.3.x and version 2.x. Since the configuration of these two differ quite substantially in some places, we will focus this course on the configuration of version 2. Should you still be running version 1.3, we suggest you upgrade. You'll need to do it at some point, so the sooner the better (IMHO). If you can't, most of this course will apply to you, but your mileage may

vary somewhat.

Basic server design

The Apache web server has been designed to be used in either a modular or non-modular way. In the former, modules are compiled separately from the core Apache server, and loaded dynamically as they are needed. This, as you can imagine, is very useful as it keeps the core software to a minimum, thereby using less operating system resources. The downsides of course are that the initial startup of the server is a little slower, and the overhead in loading a module adds to the latency in offering web pages. The latter means of compiling Apache is to include all the "modules" in the core code. This means that Apache has a bigger "footprint", but the latency overhead is down to the bare minimum. Which means you use is based upon personal choice and load on you web server. Generally though, when you unpack an Apache that has been pre-compiled (i.e. It's already in a .deb or .rpm package format), it is compiled to be modular. This is how we'll use it for this course.

Basic configuration

The core Apache server is configured using one text configuration file - **`httpd.conf`**. This usually resides in **`/etc/httpd`**, but may be elsewhere depending on your distribution. The **`httpd.conf`** file is fairly well documented, however there are additional documentation with Apache that is an excellent resource to keep handy.

The server has 3 sections to the configuration file:

1. The global configuration settings
2. The main server configuration settings
3. The virtual hosts

Global configuration settings

In this part of the configuration file, the settings affect the overall operation of the server. Setting such as the minimum number of servers to start, the maximum number of servers to start, the server root directory and what port to listen on for http requests (the default port is 80, although you may make this whatever you wish).

Main server configuration settings

The majority of the server configuration happens within this section of the file. This is where we specify the `DocumentRoot`, the place we put our web pages that we want served to the public. This is where permissions for accessing the directories are defined and where authentication is configured.

The virtual hosts section

Hosting of many sites does not require many servers. Apache has the ability to divide it's time by offering web pages for different web sites. The web site www.QEDux.co.za, is hosted on the same web server as www.hamishwhittal.org.za. Apache is operating as a virtual host - it's offering two sites from a single server. We" configure some virtual hosts later in the course.

We're out of the starting blocks

So, let's begin by configuring a simple web server, creating a web page or two and build on our knowledge from there.

In it's simplest form, Apache is quite easy to configure. Freshly unwrapped out of the plastic, it is almost configured. Just a couple of tweaks and we're off.

The Listen directive

Directives are keywords that are used in the configuration file. The listen directive for example would tell the server what port it will be listening on.

```
Listen 80
```

Will listen on port 80, the default web server port. Directives are case insensitive, so "Listen" and "listen" will be the same directive.

The ServerAdmin directive

This is the email address of the administrator of this web server. Generally, web administrators use some generic name such as webmaster@QEDux.co.za, rather than their own name. I suggest you follow that convention. How do you get this email relayed to your personal email address? Create an alias on the mail server (which is covered in the configuration of the MTA).

In my case, I've set mine as follows:

```

```

```
ServerAdmin webmistress@QEDux.co.za
```

The DocumentRoot directive

Finally, to get us going, we need to verify our `DocumentRoot` directive. This dictates where out actual .html pages will reside. The `DocumentRoot` on my machine is:

```
DocumentRoot /var/www/html
```

Since this where the html files reside, you will need to put a file in this directory called **`index.html`**. The **`index.html`** file is attached to this course and is called **`index-first.html`**. Not only will you will need to copy it to your `DocumentRoot` directory but you will also be required to rename it to **`index.html`**. When a client enters the URL in their browser as follows:

```
http://www.QEDux.co.za/
```

they expect to get the **`index.html`** page in return. The web server in turn, appends the "/" to the end of the `DocumentRoot` in order to find the page in question. This, in essence is similar to a chroot in Linux. When you chroot, you "hide" the real path to the file system, presenting the user with what looks like the root directory. Here the same applies. You would not want your clients to be able to see that the real directory lives in **`/var/www/html`**, they should just see it a "/"

Starting Apache

Starting Apache can be done from the command line. However before you jump in boots and all, it might be wise to run your **`httpd.conf`** file through the syntax checker. Start by getting help on the httpd options:

```
httpd -?
```

Once you've scanned the options (and of course read the man page for httpd ;-), you can run the syntax checker using the switch:

```
httpd -t
```

```
```

which should tell you whether you've made any glaring errors in the **`httpd.conf`** file. If not, we're in business. Now run:

```
httpd
```

This will pause briefly on the command line and then return you to the prompt. Now, it is worth looking at the httpd process BEFORE looking at the web page (**`index.html`**) you copied to the `DocumentRoot` earlier. To do this type:

```
ps -ef|grep httpd
```

In my case, I see:

```
root      1983     1  2 22:36 ?        00:00:00 httpd
```

Now, using a text browser like links or lynx, type the following command:

```
lynx http://your-IP-address/index.html
```

Bingo, the page is served to you. If you don't have lynx or links (both text based browsers) then fire up your old trusty GUI browser and type in the address.

Now, we need to look back at the httpd processes running on your system after you've retrieved the web page. Again type:

```
ps -ef|grep httpd
```

Now, in addition to the one server owned by root, another stack of servers are running, all being owned (in my case) by Apache.

```
apache    1791  1788  0 10:44 ?        00:00:00 /usr/sbin/httpd
apache    1792  1788  0 10:44 ?        00:00:00 /usr/sbin/httpd
......
```

```
apache    1798  1788  0 10:44 ?        00:00:00 /usr/sbin/httpd
```

Why is this? Why are there many servers started when you purposely only started one?

Multi-processing Modules (MPM)

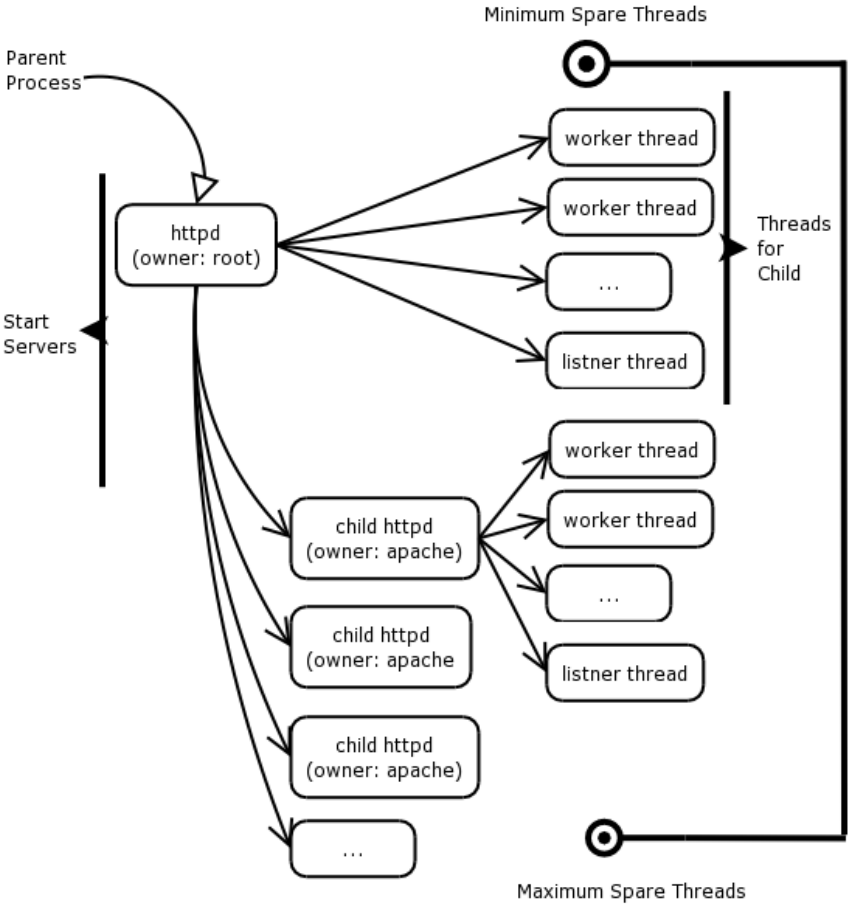
Apache has been designed to run using child processes and threading. In essence, these features make it possible for Apache to service incoming client requests in a timely and efficient manner. It is important to understand the difference between threading and child processes. While this is not essential to the basic configuration of Apache it will help you understand how Apache is servicing the client requests.

Apache uses two sets of modules to address these modes of operation, namely:

1. the worker module for the threading and
2. the prefork module for the handling of the child process

The diagram below is a schematic showing the worker module in action.

Figure 1.1. The Apache Worker Module



It works as follows:

The workers

The following description should be read in conjunction with the diagram above. The root user starts the initial process, in order that Apache can use a low numbered port (80). Thereafter, this root-owned process forks a number of child processes. Each child process starts a number of threads (`ThreadsPerChild` directive) to answer client requests. Child processes are started or killed in order to keep the number of threads within the boundaries specified in `MinSpareThreads` and `MaxSpareThreads`. Every child forks a single listener thread which will accept incoming requests and forward them to a worker thread to be answered. Other parameters used in the worker.c directive are:

`MaxClients`: This is the maximum number of clients that can be served by all processes and threads. Note that the maximum active child processes is obtained by:

```
MaxClients / ThreadsPerChild
```

`ServerLimit` is the hard limit of the number of active child processes. This should always be greater than:

```
MaxClients / ThreadsPerChild
```

Finally, `ThreadLimit` is the hard limit of the number of threads. This should also be larger than `ThreadsPerChild`

All this is bounded by the `IfModule worker.c </IfModule>` directives. On the whole, you probably will not need to modify these settings, but at least you know what they are for now.

The child processes

Handling requests using the prefork module is similar in many respects to the worker module, the main difference being that this is not a threading modules, meaning that threads do not answer the client http requests, the child processes themselves do. Again, root starts the initial httpd process in order to bind to a low-numbered port.

The startup process then starts a number of child processes (`StartServers`) to answer incoming requests. Since children in the prefork module do not use threads, they have to answer the requests themselves. Apache will regulate the number of servers. `MinSpareServers` is the minimum number of spare servers that will be running at all times, with the upper limit bounded by `MaxSpareServers`. The `MaxClients` will be the maximum number of client requests that may be answered at one time. Finally, the `MaxRequestsPerChild` is the number of requests that a single child process can handle. All these are configurable within the `IfModule`prefork.c`</IfModule>` directives, however the defaults will probably suite you for the purpose of this course and probably for a server in the office too.

Http ports

By default, web servers will mostly listen on port 80. While this is not a prerequisite, the browsers almost without fail will try to connect to a web server on port 80. So, the two need to co-encide - naturally. Linux ports below 1024 are considered well-known and only root can bind to these ports from the operating systems level. Hence the need to start Apache as root, and thereafter switching to an Apache (or other) user. As of Apache 2, a port must be specified within the configuration file as defined by the Listen directive. By default, Apache will offer web pages on every IP address on the server on this port. In some instances, this may not be the desired

operation. Thus the Listen directive can also take an IP address and a port number as follows:

```
Listen 10.0.0.2:1080
Listen 192.168.1.144:80
```

Here the server contains two host addresses each answering http requests on different ports.

Modularity of Apache

As mentioned earlier, Apache can be used as a monolithic server (all modules required are compiled into the executable), or in a modular fashion using dynamic shared objects (DSO's). The latter is the preferred means of using Apache as the server has a smaller memory footprint. we'll discuss the DSO method of configuring and using Apache, and leave it up to you, the learner, to read the documentation if you wish to do anything different from this.

Modules are loaded into the core server using the mod_so module. In order to see what modules are compiled into Apache running on your machine, you can issue the command (on the command line):

```
httpd -l
```

On my machine, this command returns:

```
linux:/ # httpd -l
Compiled-in modules:
  core.c
  prefork.c
  http_core.c
  mod_so.c
```

Clearly, from our discussion earlier on worker versus prefork modules, my Apache will use the prefork directives. Also, in order for Apache to be able to load modules as required, it needs the mod_so compiled in too. If it was not compiled in, it would be unable to load itself, never mind any other modules!

The `LoadModule` directive will load the associated module when needed. The syntax for this directive is straight-forward.

```
LoadModule foo_module modules/foo_mod.so
```

This will load foo_module from the file in the modules directory called **foo_mod.so**.

Once this is configured, there's little else to do. There are however, configuration files that may be associated with each module. PHP for example has a configuration file to specify run-time configuration directives. Configuration files usually reside in the same (or similar) place as your **httpd.conf** file described earlier. On my system, these files reside in:

```
/etc/httpd/conf.d/
```

Included below is the php.conf file from this directory.

```
#
# PHP is an HTML-embedded scripting language which attempts to
# make it easy for developers to write dynamically generated
# webpages.
#

LoadModule php4_module modules/libphp4.so

#
# Cause the PHP interpreter handle files with a .php extension.
#
<Files *.php>
    SetOutputFilter PHP
    SetInputFilter PHP
    LimitRequestBody 524288
</Files>

#
# Add index.php to the list of files that will be served
# as directory indexes.
#
DirectoryIndex index.php
```

This configuration file is "appended" to the **httpd.conf** file when the **mod_php.so** is loaded. You may notice that the structure of this file looks very similar to the Apache configuration file - for good reason.

Documentation of the modules is generally shipped with the Apache manual, which will reside on the default web site on the web server. In my case, my default

`DocumentRoot` is **`/var/www/html`**, so the manual for Apache resides in **`/var/www/manual`**, and the module manuals reside in **`/var/www/manual`**/mod/.

We'll cover the use of some of these modules during the course. For now though, accept that the modules can be used for a variety of purposes.

User and group information

As mentioned earlier, Apache will run as root on startup (if you want to use low-numbered ports). Once this is done though, the server will start child process as the user/group specified in the server configuration file. The `User` and `Group` directives specify this.

```
User <your Apache user here>
Group <your Apache group here>
```

Starting child processes as this user will have an effect on your CGI scripts running later. We'll cover CGI and a program called suexec later in the course, but it is worth remembering this fact.

ServerName

Often a server may not be named the same as the host you wish your users to refer to it as. For example, my server in my office is called ipenguini.QEDux.co.za, while I wish users wanting to contact my web site to use www.QEDux.co.za. Since these are in fact the same server (using the CNAME Internet Resource Record in the DNS), I would put my `ServerName` as www.QEDux.co.za.

DocumentRoot

This directive is essential to the running of Apache. It dictates where the contents on the web site will reside. Assume that users wish to contact my web site www.QEDux.co.za. They will type in the following URL:

```
http://www.QEDux.co.za/index.html
```

Clearly, they could have left out the **`index.html`**, but I have put it in here for a reason. Now, presumably, your **`index.html`** file does not reside in the root directory. In my case it lives in:

```
/var/www/QEDux.co.za/html
```

So my `DocumentRoot` directory is set to:

```
DocumentRoot /var/www/QEDux.co.za/html
```

The `DocumentRoot` directive translates in the URL to the "Apache root directory /", hence the "/" on the end of the URL

```
http://www.QEDux.co.za/
```

We wouldn't want users to type in:

```
http://www.QEDux.co.za/var/www/QEDux.co.za/html/index.html
```

Apart from this presenting a security problem, it would confuse the users no end.

In sum then, the `DocumentRoot` directory specifies the point of departure for the **`index.html`** page and the rest of the web site. In essence, it "hides" the **`/var/www/QEDux.co.za/html`**, parading it as "/" to the user.

OK. With all this new-found knowledge, you're wanting to get cracking with your new web server, but wait. There's more. While we have defined the `DocumentRoot` and the `ServerName`, we still need to define what operations can be performed in directories within the `DocumentRoot`.

In order to do this, Apache has "container" blocks. These blocks are very XML-ish in nature. For example, there is a `Directory` container, that contains information about a particular directory. The `Directory` container is closed using a `</Directory>` container directive. Other containers include `DirectoryMatch`, `Files`, `FilesMatch`, etc. The general format of a container is:

```
<Directory>
        Some options and directives go here
        Some more options and directives
</Directory>
```

By default, we need to specify a really secure set of directives for the root ("/") directory. [1]

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

Without going in to too much detail here (it will be covered later), we allow users to follow symbolic links from the root directory, but allow no other options to override the options in this container. Options can be overridden using a **`.htaccess`** file (described later) if the `AllowOverride` was set to "All".

Once we've set a very restrictive set of permission for the root directory, we set up the permission for the `DocumentRoot` directory.

```
<Directory "/var/www/QEDux.co.za/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Here, the options are a little more relaxed. First, we allow users to follow symbolic links. Also, if they happen upon a directory without an **`index.html`** or **`default.html`** file, they will obtain an index of the files in the directory - in a similar way you get an index of your directory if you type:

```
file:///var/log
```

into your browser window. We still disallow any directives set in a **`.htaccess`** file from overriding the directives set in this container, and finally we set access to this directory. In this example, we will check the Allow list, and then the deny list of whether this user can gain access. This may be particularly helpful when restricting web sites to say, the help desk operators, or the call-centre people. In the example, we first check the allow list (which admits everyone) and then the deny list (which is absent here), and allow or deny people accordingly.

After setting your `Directory` container symbol (adjust to your `DocumentRoot`), you are ready to start your web server. In order to do this, you will probably want to

[1] don't get confused between the `DocumentRoot` and the real root directory. Here, we are referring to the real file system root directory.

create a default web page. I have included one for download with this course - **`index-first.html`**. Rename it to **`index.html`** and put it in your `DocumentRoot` directory. Point your browser at your machine and voila, you should see the web page.

This is an example of the directory container for the "cgi-bin" directory. Note how the directory is specified.

```
<Directory "/srv/www/cgi-bin">
        AllowOverride None
        Options +ExecCGI -Includes
        Order allow,deny
        Allow from all
</Directory>
```

In order to use your machine name rather than your IP address in the URL, you will have to modify your hosts file to contain the FQDN of your hosts. My host for example is defender.QEDux.co.za, so I put www.QEDux.co.za as an alias in my hosts file as follows:

```
192.168.1.24 defender.QEDux.co.za www.QEDux.co.za
```

Now that you have got the basic thing going, it's time to return to those options in the containers. Options are controlled using a "+" or a "-" in front of the option. No sign preceding an option is taken to mean a "+". In our preceding examples, the option FollowSymLinks was not preceded by a "+", but the "+" was implied. A subtlety here is that if the option is preceded by a "+", the option will be added to any previous options as laid out elsewhere. So the options may be:

- All: Allow all options, all that is except MultiViews.
- FollowSymLinks: Follow symbolic links from this directory, even if the target directory does not actually reside in the same `DocumentRoot` directory.
- SymLinksIfOwnerMatch: Follow the symbolic links, but only if the owner of the symbolic links matches the owner of the directory in which the file currently resides.
- ExecCGI: Allow execution of CGI scripts in this directory
- Includes: Allow server side includes. SSI will be covered in more detail later in the course.

- Indexes: If the directory in which the user finds themselves does not contain any **`index.html`** or **`default.html`** (as specified by the directive DirectoryIndex), then a listing (probably with pretty icons) will be displayed.

- MultiViews: This is a little more complex. Web sites can be tailored in a number of ways. They could offer your website in Xhosa for example, and the users home language would be selected automatically depending on their browser preferences. My index page may be index.xh.html as well as index.en.html. Now when a native English speaker views my site (with their browser configured appropriately), they will see the English page, while when the native Xhosa speaker views the site (browser configured correctly again), they will see the Xhosa version. Allowing MultiViews makes this possible. Language support is not the only thing available in MultiViews. Text/html versus text only, jpg or gif versus png's can also be selected based upon priorities.

The example below, taken from the Apache manual shows that text/html is preferred over straight text and gif's and jpg's are preferred over all other image types.

```
Accept: text/html; q=1.0, text/*; q=0.8, image/gif; q=0.6,
        image/jpeg; q=0.6, image/*; q=0.5, */*; q=0.1
```

Since MultiViews are a directory based directive, they need to be enabled using the **`.htaccess`** files in a directory in order to take effect (see a description of the **`.htaccess`** file below).

let's return now to the `AllowOverride` directive.

- The `AllowOverride` directive is only allowed in a `Directory` container.

- As mentioned earlier when discussing the "/" (root) directory, when this directive is set to "None" the **`.htaccess`** files are completely ignored.

- When set to "All" the directives in the **`.htaccess`** files are allowed based upon their "Context".

Context refers to the context (or containers) in which directives are allowed or denied. In some contexts, directives are not allowed due to possible breaches of security, or because they don't make much sense in the context

Other override directives include:

- authentication directives (AuthConfig - see mod_auth for more information),
- file information directives (FileInfo - see mod_mime for more information), directory indexing (Indexes - see mod_autoindex for more information),
- limitation of access to the web site (Limit - see mod_access for more information) or,
- any of the options described above - provided of course they make sense in the context of the container (Options)

Using these options in, say, a **`.htaccess`** file in some web directory will allow us to limit access to that directory to individuals in a certain subnet, or show the indexes in a particular manner, etc.

The final piece of our container pie above relates to access to this page or site. This all falls under the mod_access module. mod_access is responsible for controlling access to the page/site based upon one, or a number of criteria:

- Clients hostname
- Clients IP address and/or subnet
- Environmental variables being set

The last of these is too complex for this course and will not be discussed now. The other two are relatively easy.

We may set an allow/deny directive as follows:

```
Allow from all
```

which will naturally allow for access from all clients. Alternatively:

```
Allow from 172.16.1.0/24
```

will allow only users on the network 172.16.1.0 to connect to the web site. Others examples:

```
Allow from QEDux.co.za
```

```
```

Only allow clients to connect from the domain QEDux.co.za.

```
Allow from \
  172.16.1.1 ipenguini.QEDux.co.za 192.168.10.0/255.255.255.0
```

Only allow clients on the 192.168.10.x network to connect, as well as the hosts ipenguini.QEDux.co.za and 172.16.1.1.

The deny rules work identically. The only outstanding thing is to determine the order in which these rules are applied. For that the `Order` directive is used - simply indicating the order in which the allow and deny directives are applied.

Assuming I have the directives:

```
Allow from \
  172.16.1.1 ipenguini.QEDux.co.za 192.168.10.0/255.255.255.0
Deny from all
Order Allow,Deny
```

Once a match is made, no further checking is done. So, had I switched the `Order` directive to:

```
Order Deny,Allow
```

Then even the poor souls that appear in the allow directive would not be allowed through as the deny is denying everyone at the outset.

Virtual hosts

Apache is able to serve a number of different web-sites from the same machine. The way this is achieved is through the use of virtual hosts in the configuration file. It should be noted that if even a single virtual host is described, all containers outside the `VirtualHosts` container will be ignored. Thus, if you set up even a single virtual host, then your original web site must fall within the VirtualHosts container too.

There are two types of virtual hosts:

1. name based virtual hosts and,

2. IP based virtual hosts.

IP based virtual hosts require separate IP addresses in order to run, while name-based ones run off the same IP address, but are called different things (perhaps gadgets.co.za and widgets.co.za). We will look at name-based ones in this course. For IP based ones, consult the Apache documentation.

The first directive required is the "`NameVirtualHost`". This directive is used by the hosts to be associated with a single IP address. Remember, in name-based virtual hosts, there may be multiple names on a single IP address. Within the `VirtualHosts` container, we can include all the directives we used without virtual hosts.

An example is included below:

```
#the widgets website
NameVirtualHost 192.168.0.211
<VirtualHost 192.168.0.211>
   ServerAdmin webmaster@widgets.co.za
   DocumentRoot /var/www/widgets.co.za/html
   ServerName www.QEDux.co.za
   ErrorLog logs/www.widgets.co.za-error_log
   CustomLog logs/www.widgets.co.za-access_log common

     <Directory "/var/www/widgets.co.za/html">
        Options Indexes +Includes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
     </Directory>

     Alias /icons/ "/var/www/widgets.co.za/icons/"
     <Directory "/var/www/widgets.co.za/icons">
        Options Indexes MultiViews
        AllowOverride None
        Order allow,deny
        Allow from all
     </Directory>

    ScriptAlias /cgi-bin/ "/var/www/widgets.co.za/ \
        cgi-bin/"
    #
    # "/var/www/cgi-bin" should be changed to
    # whatever your ScriptAliased
    # CGI directory exists, if you have that configured.
    #
    <Directory "/var/www/widgets.co.za/cgi-bin">
       AllowOverride None
       Options None
       Order allow,deny
```

```
      Allow from all
    </Directory>

</VirtualHost>

# The gadgets web site.
<VirtualHost 192.168.0.211>
   ServerAdmin webmaster@gadgets.co.za
   DocumentRoot /var/www/gadgets.co.za/html
   ServerName www.QEDux.co.za
   ErrorLog logs/www.gadgets.co.za-error_log
   CustomLog logs/www.gadgets.co.za-access_log common

     <Directory "/var/www/gadgets.co.za/html">
        Options Indexes +Includes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
     </Directory>

     Alias /icons/ "/var/www/gadgets.co.za/icons/"
     <Directory "/var/www/gadgets.co.za/icons">
        Options Indexes MultiViews
        AllowOverride None
        Order allow,deny
        Allow from all
     </Directory>

    ScriptAlias /cgi-bin/ "/var/www/gadgets.co.za/cgi-bin/"
    #
    # "/var/www/cgi-bin" should be changed to
    # whatever your ScriptAliased
    # CGI directory exists, if you have that configured.
    #
    <Directory "/var/www/gadgets.co.za/cgi-bin">
       AllowOverride None
       Options None
       Order allow,deny
       Allow from all
    </Directory>

</VirtualHost>
```

As you can see, the widgets and the gadgets web sites are hosted on the same web server. A client pointing their browser to widgets.co.za would be directed to the widgets web site, while the gadgets.co.za This is virtual hosting. It can get more complex than this, but for now, this is adequate. Obviously the more virtual hosts you require, the more `VirtualHosts` containers you will require too.

Common Gateway Interface (CGI)

CGI is handled in Apache using a program called suexec. This program is used to run applications from within Apache. Naturally, this is a fairly dangerous application

to allow anyone access to use. A simple CGI script, with malicious intent can wreak havoc on you system. It is wise to keep tight control over the CGI that is allowed to run on your system. CGI is quite simple really. It is a piece of source code (such as perl, C, Java, etc.) that runs, and creates output that in HTML format. let's look at a simple CGI program using the shell:

```
#!/bin/bash
echo "Content-type: text/html\n\n"
echo "<HTML>\n"
echo "<HEAD><TITLE>My First CGI Script</TITLE></HEAD>\n"
echo "<BODY> \
    <H1>About my first automatically generated HTML code \
    </H1>\n"
echo "<HR/>\n"
echo "This is a really cool way to generate HTML<br/>"
print "</BODY></HTML>\n"
exit (0)
```

Putting this in a file, changing it's mode, and then pointing your web browser to it will cause it to run. The output is sent back to the web browser. CGI gets pretty complex and this is by no means a CGI tutorial, but simple CGI can be very effective in delivering information to the user.

A directive "`ScriptAlias`" indicates where CGI scripts can be placed in the directory hierarchy. In the case of the example below, the scripts are placed in the **`/var/www/widgets.co.za/cgi-bin`** directory.

```
ScriptAlias /cgi-bin/ "/var/www/widgets.co.za/cgi-bin/"
<Directory "/var/www/widgets.co.za/cgi-bin">
   AllowOverride None
   Options None
   Order allow,deny
   Allow from all
</Directory>
```

Scripts placed anywhere else will simply not be allowed to be run - for obvious reasons. While CGI is still in wide use, a number of new languages have come of age. Specifically Personal Home Page (PHP) has become a very popular web authoring tool.

There is so much configuration that can be done to Apache, and taking some time to consider these options certainly does not cover it in any significant detail, but it is a start.

Chapter 2. Berkley Internet Name Daemons (BIND)

Introduction

This module will walk you through the configuration of a DNS name server. In the Linux world, DNS is generally maintained by the BIND software, which is currently in it's 9th revision. There are, of course other DNS server software out there, but since almost the entire Internet uses BIND, and it has been the most long-serving DNS server, we will focus on BIND in this course.[2]

Before we being diving into configuring BIND, we need to review a couple of topics that you may or may not know about. We will also need to sketch the setup of our private network to make understanding the structure of our DNS simpler.

Our sample network

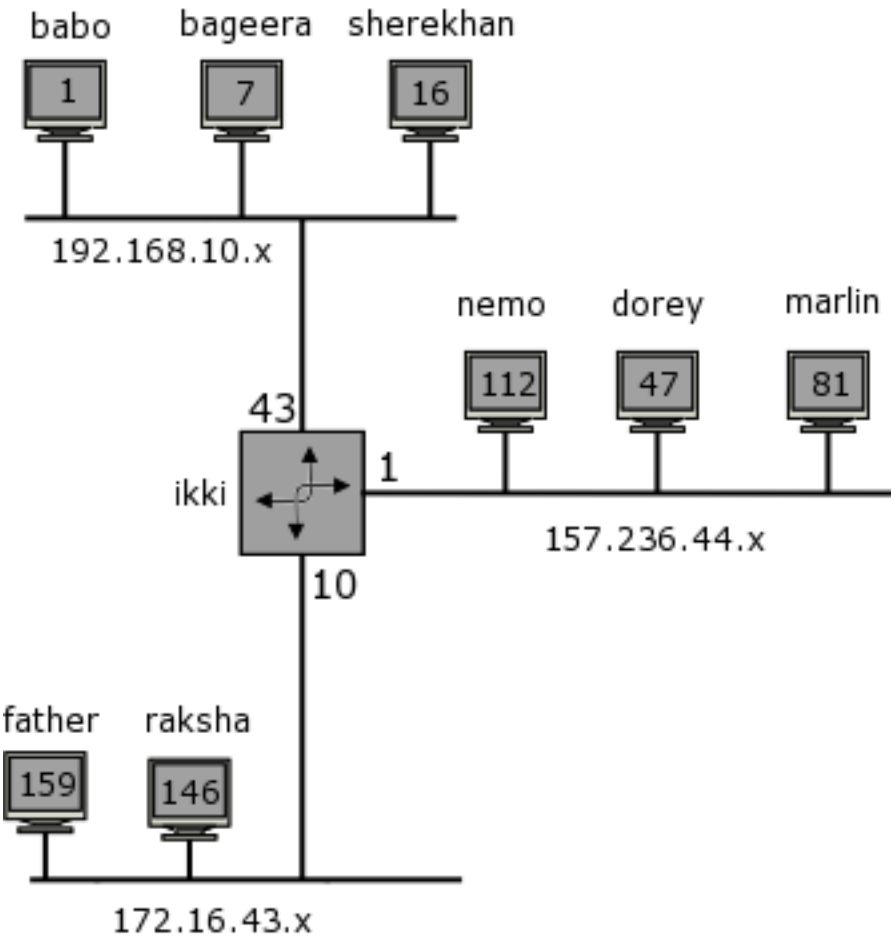
In your class, (or even if you're studying this course at home) you may not have all the machines listed in this sample network layout. don't worry. Neither did I. You see, getting something listed in the DNS does not necessarily mean the machine has to be present physically, since DNS is merely a service that will translate IP addresses to names or visa versa.

There are many domains on the Internet that have been reserved, but are currently not in use. In this module, we're going to create a fictitious domain, and while we're about it, we'll create a couple of fictitious machines too. The minimum number of machines you will need to get this course to work is 2, which is the minimum required in the course specification.

Figure 2.1 [21] below illustrates the proposed configuration of our sample network. You should see that some of the hosts are multi-homed. Multihomed hosts are those that are connected to two networks simultaneously.

Figure 2.1. Our sample network

[2]The other DNS server that is becoming popular is Dan Bernstein's djbdns DNS server. See http://djbdns.org for more information on this DNS server alternative.



A review of some theory on domains and sub-domains

The Internet is comprised of domains, organized into a hierarchical structure. At the top of this hierarchy is the root domain. In design, the domain hierarchy is similar to the UNIX/Linux file system structure.

Figure 2.2 [23] illustrates a typical UNIX file system structure, while Figure 2.3 [23] shows a comparative diagram of the domain name hierarchy.

As you may have guessed, a domain name is written from most specific part to least specific part. Thus, the domain QEDux.co.za has a host called www, and another called FTP, while the domain co.za had many sub-domains, of which QEDux is one.

Another may be pumphaus.co.za. So as we read the domain from left to right, we read from most specific to least specific, "za" being much less specific about a host than "www".

Figure 2.2. Unix Filesystem hierarchy

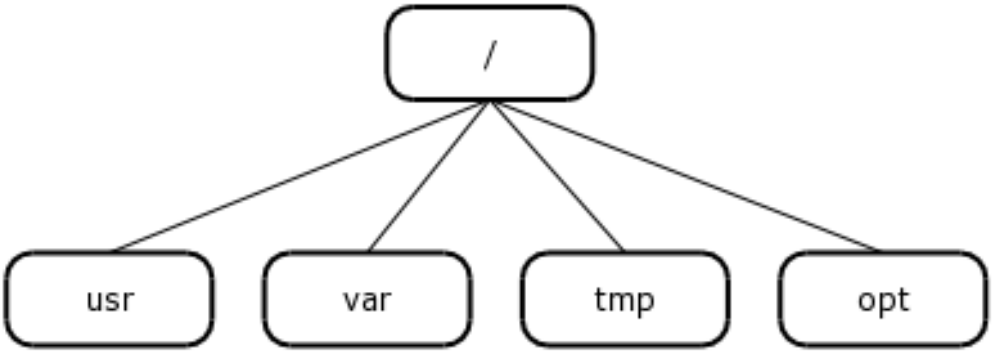
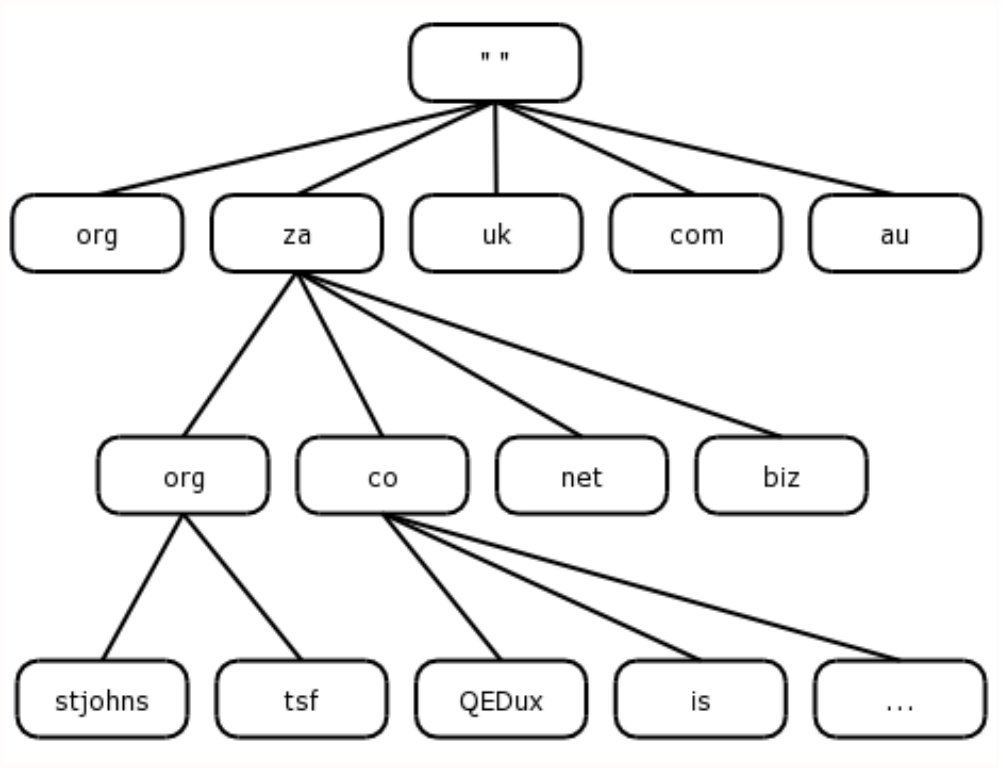


Figure 2.3. Domain name hierarchy



What is often not shown in a fully qualified domain name (FQDN) is the root node. Why?

Well the root node is actually "" (null) and the separator is the full-stop (.), so FQDN's should actually be written as:

```
"www"."QEDux"."co"."za".""
```

Where I have included the quotes ("") purely as a means of showing the root node on the end. We would not really write a FQDN like this. It would instead be written as: www.QEDux.co.za

Notice that the trailing dot is dropped, as the root node is not shown.

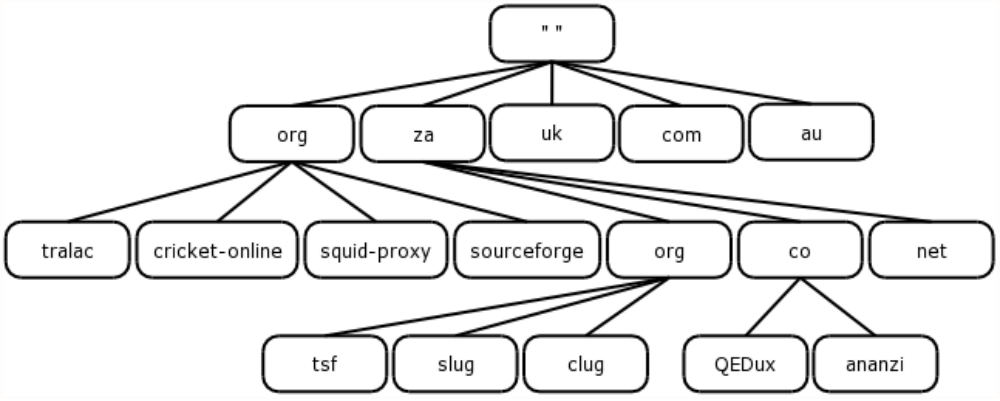
Within our domain hierarchy, sibling nodes MUST have unique names. Take the "org" domain for example, at tier one of the domain hierarchy, there is an "org" sub-domain and some examples may include "tralac.org", "glug.org", while within the "za" sub-domain, we can once again have the "org" domain.

This time however, the sub-domain falls within the "za" domain, making "tsf.org.za" a distinctly different domain from "tsf.org". Similarly, "tsf.org.za" would be unable to have two sub-domains within their domain called "slug".

If one were to extrapolate back to the analogy of the UNIX/Linux file system, we are able to have two "sbin" directories, however one "hangs off" /usr, while the other "hangs off" the root directory (/).

To flog this horse, there are MANY hosts on the Internet, which have the name "www", but because they are all contained within their domains, our packets traversing the network will know which one we mean to make contact with.

Figure 2.4. Domains



DNS - Administration and delegation

OK, we have now reviewed domains and sub-domains, next we will deal with how to administer DNS?

Is there some person responsible for the Internet DNS as a whole? No. There is an organization that is responsible for administering the use of domain names (and IP addresses too), but they could not possibly be responsible to maintaining the DNS for the entire Internet.

Apart from being a completely unmanageable job, it would also be very prone to breaking due to this central point of failure.

As a result, there is a system of delegation. In much the same way that all the computers may belong to your school, or company, while only a couple of machines are delegated as your responsibility, DNS is controlled by ICANN (Internet Corporation for Assigned Names and Numbers), and authority for your domain may

well be assigned to your organization, ICANN having little to do with it's administration.

In order to manage the DNS hierarchy, ICANN delegate responsibility for your domain to another authority. This may be your organization, or, if you're too small or lack the technical expertise, it may be your Internet Service Provider (ISP). In my case, for example, my ISP administers my DNS, not because I lack the technical know-how, but due to the small number of hosts I have on my network, it would be more effort than it would be worth.

let's look at an example. Supposing my company were large and I control my DNS, I could subdivide my domain further. Perhaps I want to add two new sub-domains. Since sales people never listen to technical people ;-), I decide that I would want two new domains:

```
sales.QEDux.co.za and
tech.QEDux.co.za
```

Now we could have two web servers serving information relating to the sales department and the technical department.

Each server could be called "www", without fear of a clash of host names.

Since we have authority over this domain, we can simply configure our DNS to handle the new sub-domains without having to contact ICANN again.

This raises an important but subtle issue: that of domains and zones. We are learning in this module how to configure a name server (BIND being the software offering this service).

Name servers have complete "knowledge about" and "authority over" the sub-domain they have jurisdiction over and this sub-domain is referred to as a zone.

Some Examples of Zones

Thus if I start a name server containing all records for the domain QEDux.co.za, my name server will be said to have authority over the QEDux.co.za zone.

As a more complex example, the "gov.za" domain (which is actually a sub-domain of the "za" domain) may delegate authority for each of the regional governments. In this way, the City of Cape Town is responsible for their domain (capetown.gov.za) and the City of Johannesburg is responsible for their domain (joburg.gov.za).

Each of these regional governments have a "zone" for which they need to keep records. It would make no sense for the City of Cape Town to try to keep records for

the City of Johannesburg. By delegating these responsibilities, the individuals and machines responsible for the "gov.za" domain only need keep records of who has responsibility for the capetown and joburg domains. They trust (rightly or wrongly) that Cape Town and Johannesburg will have the savvy to manage their own domain.

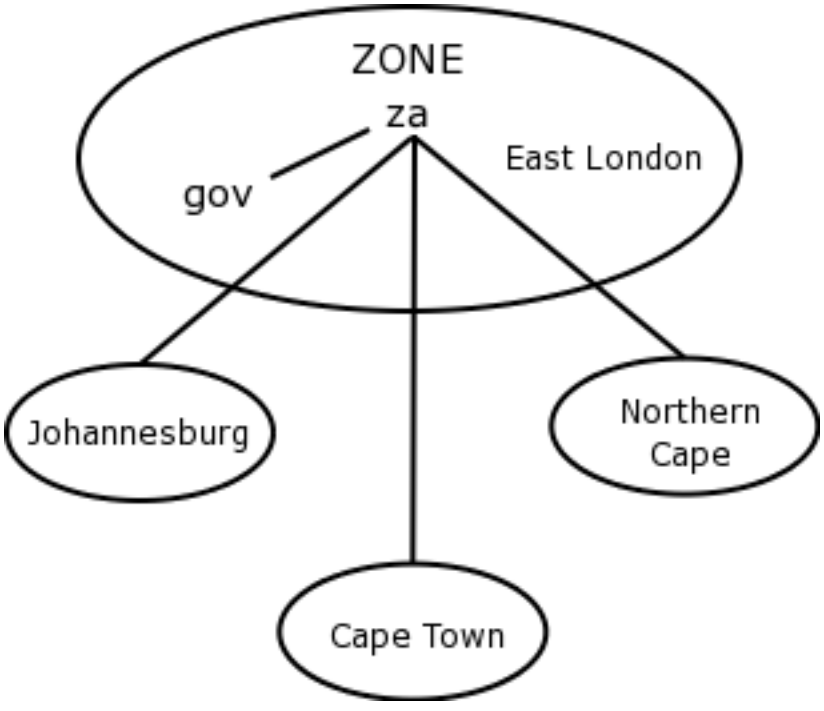
Zones in Summary

In sum then, a zone is a sub-domain for which a name server has a complete set of records (commonly these will be stored in a file on the name server).

What complicates this picture is the fact that some of the regional authorities may not have sufficient skills to maintain their own domain. In this case, the authorities responsible for the "gov.za" domain may administer this domain too. As a result, the zone for the "gov.za" will now include this regional information too.

The zone is thus not restricted to a domain. Figure 2.5 [27] illustrates the zones for this hypothetical gov.za domain.

Figure 2.5. gov.za zones



Name resolution

For the sake of completeness, we will review the process of domain name resolution.

Resolvers (the client side software) is, on the whole, not particularly intelligent. As a result of this, the name server must be able to provide information for domains for which they are authorities, and the name server should also have the ability to search within the name space (all the domains from the root) for information about hosts for which they are not authoritative.

No single name server could possibly hold records for every domain on the Internet. In order to have a point to begin searching the name space, there are a number of root name servers. These servers fulfill the role of containing information about who the authorities for particular domains are. In much the same was that the authority for the "gov.za" domain can not answer queries about what the IP address for host www.capetown.gov.za is, but instead passed the query on to the authority for the capetown.gov.za domain, the root name servers contain information about who to forward the query to.

In effect, the root servers contain only a relatively small number of hosts for which they know the answers (relative that is, when compared to the number or hosts and domains on the Internet as a whole). let's go on a virtual travel trip. Been to Antarctica lately? Go to http://www.aad.gov.au (and while we're there, head for ..... the webcam at www.aad.gov.au/asset/webcams/mawson/default.asp)

Perhaps your name server may have this domain in its local cache, but let's assume for now it doesn't. Since your name server has no knowledge of this domain (i.e. Your name server is non-authoritative for the aad.gov.au domain) it will request information on who the authorities are for the 'au' domain.

To do this at the command line is simple with the dig utility (or nslookup if you don't have dig).

Type:

```
dig NS au.
```

This should return the names of the servers who hold information about the Australian domains:

```
;; ANSWER SECTION:
au.         172511  IN      NS      NS.UU.NET.
au.         172511  IN      NS      MUWAYA.UCS.UNIMELB.EDU.au.
au.         172511  IN      NS      BOX2.AUNIC.NET.
au.         172511  IN      NS      SEC1.APNIC.NET.
```

```
au.         172511  IN      NS      SEC3.APNIC.NET.
au.         172511  IN      NS      ADNS1.BERKELEY.EDU.
au.         172511  IN      NS      ADNS2.BERKELEY.EDU.
```

Now query the gov.au domain.

```
       dig NS gov.au.
```

which should return:

```
;; ANSWER SECTION:
gov.au.     86400   IN      NS      ns2.ausregistry.net.
gov.au.     86400   IN      NS      ns3.ausregistry.net.
gov.au.     86400   IN      NS      ns3.melbourneit.com.
gov.au.     86400   IN      NS      ns4.ausregistry.net.
gov.au.     86400   IN      NS      ns5.ausregistry.net.
gov.au.     86400   IN      NS      box2.aunic.net.
gov.au.     86400   IN      NS      dns1.telstra.net.
gov.au.     86400   IN      NS      au2ld.csiro.au.
gov.au.     86400   IN      NS      audns.optus.net.
gov.au.     86400   IN      NS      ns1.ausregistry.net.
```

These are the name servers that hold information about the gov.au domains. You will notice that at least one of these hosts (box2.aunic.net) is repeated in both lists. To relate this back to the query issued by your name server on your resolvers behalf:

It (the name server) would have asked one of the root servers for the authoritative name servers for the 'au' domain. That received, it would then ask one of the authoritative name server for the 'gov.au' domain.

Then again for the aad.gov.za domain, which would have produced:

```
;; ANSWER SECTION:
aad.gov.au.     84727   IN      NS      alpha.aad.gov.au.
aad.gov.au.     84727   IN      NS      ns1.telstra.net.
```

Now that your name server knows whom to ask for information about www.aad.gov.au, and the fact that this is the authoritative server for this domain, it can go ahead and ask it's question:

```
dig A www.aad.gov.za
```

```
```

which should return:

```
;; ANSWER SECTION:
www.aad.gov.au.         1556    IN      A       203.39.170.21
```

Notice that I can leave off the trailing full-stop here. In fact, I could have done that from the start.

Furthermore, in the previous dig commands, we were querying for the name server (NS) records. Now, if we were to query www.aad.gov.za for its name server record, we would get nothing returned so instead, we query for the address (A) record for this host, and we get it.

Caching replies

While we are required to undergo this process for each new site we visit, if we had to do this for every site we revisit, it would be very cumbersome and slow.

Fortunately, the designers of DNS anticipated this and included a caching mechanism in the design of name servers. As a result, when we query a site for the first time, we would consult the root name server, then those for the 'au' domain, then the 'gov.au' domain, then the 'aad.gov.za' domain etc. But as these answers are returned, our name server caches them, just in case we ask again anytime soon.

Of course, we do, and now, voila our query is so much quicker since most (if not all) of the time consuming DNS querying work has been done already. we'll talk about caches later in the module when we configure our name server.

Reverse queries

we've discussed the forward query process where the names are converted into IP addresses.

we've still not discussed the reverse process - that of converting IP addresses back into names. Why would we need this functionality? Well, in the simplest case, we may wish to record in the log files of our web server, the names of the hosts that are visiting our webcam site.

Since the Internet only really talks at the network layer, the web server will only get 'hits' from an IP address. This is not very handy for the poor web administrator.

She'll simply have to give her boss a list of IP addresses of the hosts that visited, with little or no knowledge of where those visitors came from.

With reverse DNS mapping, the web server could simply look up the name of the visiting IP address and replace the IP by the name in its log files.

This would make reporting a whole lot easier and provide valuable information. let's use dig again, this time to convert an IP address into a name.

Reverse mapping is similar to name to IP mapping described above, except for the following differences:

1. The first tier domain is arpa, short for the Advanced Research Projects Agency, the people responsible for first conceiving and building the Internet.

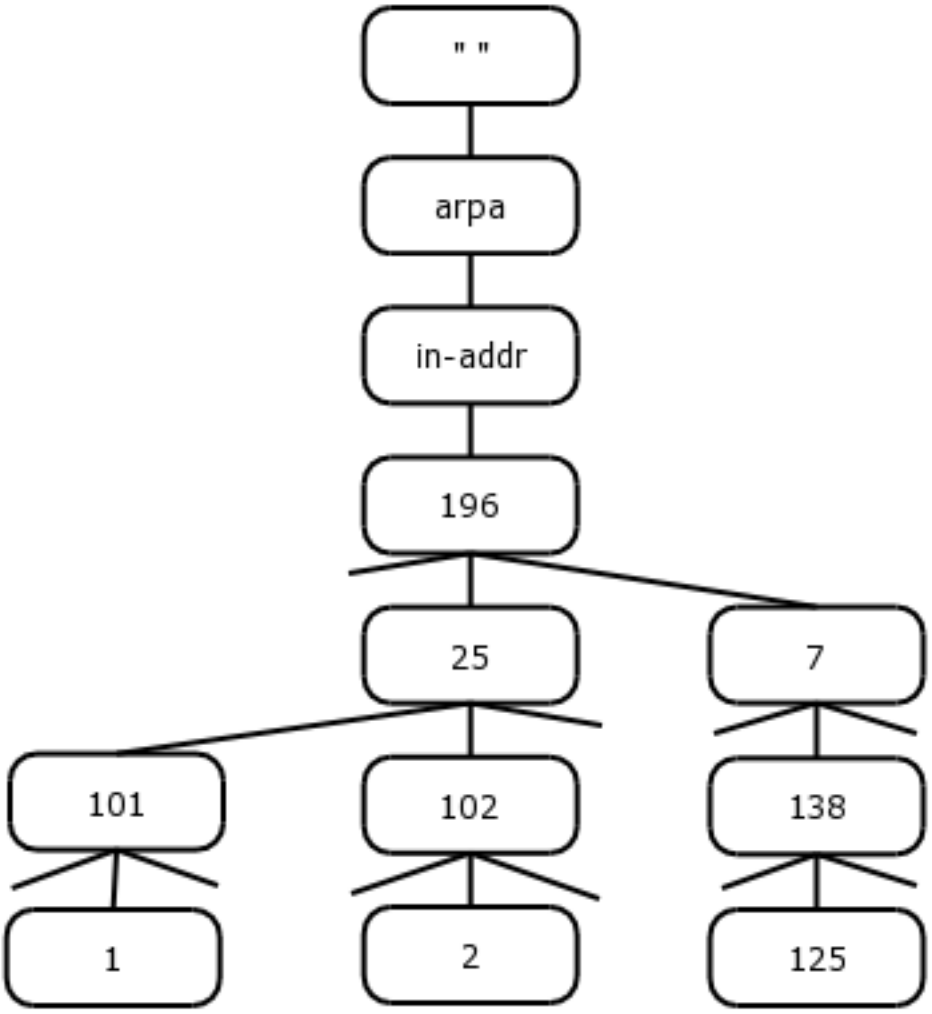
2. The second tier domain is in-addr, short for Internet Address (IP address is the term we've used till now)

Consider the IP address 196.25.102.2.

Here there is a host ".2" on a network "196.25.102". As before, the 196.25.102 is the least specific entry in this tree (there could in fact be many hosts on this network), while the most specific entry would be the host address ".2".

Refer to the Figure 2.6 [31].

Figure 2.6. Reverse Map



As before, we can reverse map this using the dig command.

Begin by typing:

```
dig in-addr.arpa NS
```

which, should return all the root servers:

```
;; ANSWER SECTION:
in-addr.arpa.    84974   IN      NS      M.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      A.ROOT-SERVERS.NET.
```

```
in-addr.arpa.    84974   IN      NS      B.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      C.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      D.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      E.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      F.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      G.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      H.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      I.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      K.ROOT-SERVERS.NET.
in-addr.arpa.    84974   IN      NS      L.ROOT-SERVERS.NET.
```

Notice that they (the root servers) all know about the in-addr.arpa domain, and so they should!

Now try:

```
dig 196.in-addr.arpa NS
```

which, should yield:

```
;; ANSWER SECTION:
196.in-addr.arpa.    84987   IN      NS      henna.ARIN.NET.
196.in-addr.arpa.    84987   IN      NS      indigo.ARIN.NET.
196.in-addr.arpa.    84987   IN      NS      epazote.ARIN.NET.
196.in-addr.arpa.    84987   IN      NS      figwort.ARIN.NET.
196.in-addr.arpa.    84987   IN      NS      ginseng.ARIN.NET.
196.in-addr.arpa.    84987   IN      NS      chia.ARIN.NET.
196.in-addr.arpa.    84987   IN      NS      dill.ARIN.NET.
```

And again:

```
dig 25.196.in-addr.arpa NS
```

yielding:

```
;; ANSWER SECTION:
25.196.in-addr.arpa.    74403   IN    NS    igubu.saix.net.
25.196.in-addr.arpa.    74403   IN    NS    sangoma.saix.net.
```

And again:

```
dig 102.25.196.in-addr.arpa NS
```

yields:

```
;; ANSWER SECTION:
102.25.196.in-addr.arpa. 31112  IN    NS  \
                quartz.mindspring.co.za.
102.25.196.in-addr.arpa. 31112  IN    NS  \
                agate.mindspring.co.za.
```

Finally,

```
dig 2.102.25.196.in-addr.arpa PTR
```

yields:

```
;; ANSWER SECTION:
2.102.25.196.in-addr.arpa. 42951 IN    PTR \
                quartz.mindspring.co.za.
```

Why did we write the IP address in reverse here?

Well, it was not really in reverse, it followed the same convention as with the name, using the most-specific part of the name first, and the least specific part at the end (www.QEDux.co.za). Referring to figure 5 again, you've just done a similar thing as with the name, only now using the IP address and ending the name with "in-addr"."arpa"."".

Masters and slaves

Due to the fact that DNS is so important in the workings of the Internet, it would make no sense to have only one name server per domain or zone.

As a result, DNS servers generally work in pairs - a master name server and a slave name server. Master name servers are, in fact, no more or less important than slave servers in their job or answering queries. Their primary difference is that the master server is the one that contains all the zone files. we'll get to set up a master shortly, so hang in there.

Since it would make no sense to maintain two copies of the zones files (one on the master and one on the slave), on startup, the slave server will do a zone transfer from the master. In effect, this obviates the need for maintaining two sets of files on two machines.

When the master server starts, it reads the zone files, and immediately begins answering queries. When the slave starts up, it transfers the zone information from the master, and only once it is complete, does it begin answering queries.

In the event that the master is not available when the slave starts up, BIND can be configured to read the zone file from a backup that was stored on the last zone transfer.

Of course, there has to be a timeout on the slaves information otherwise the slave could continue answering queries about a host long after the host has been reconfigured or decommissioned.

Configuration of the Master Server

Right, enough theory, let's get down to it.

we'll begin by configuring the master server, and once that is working fine, we'll configure the slave.

BIND has two primary areas of configuration.

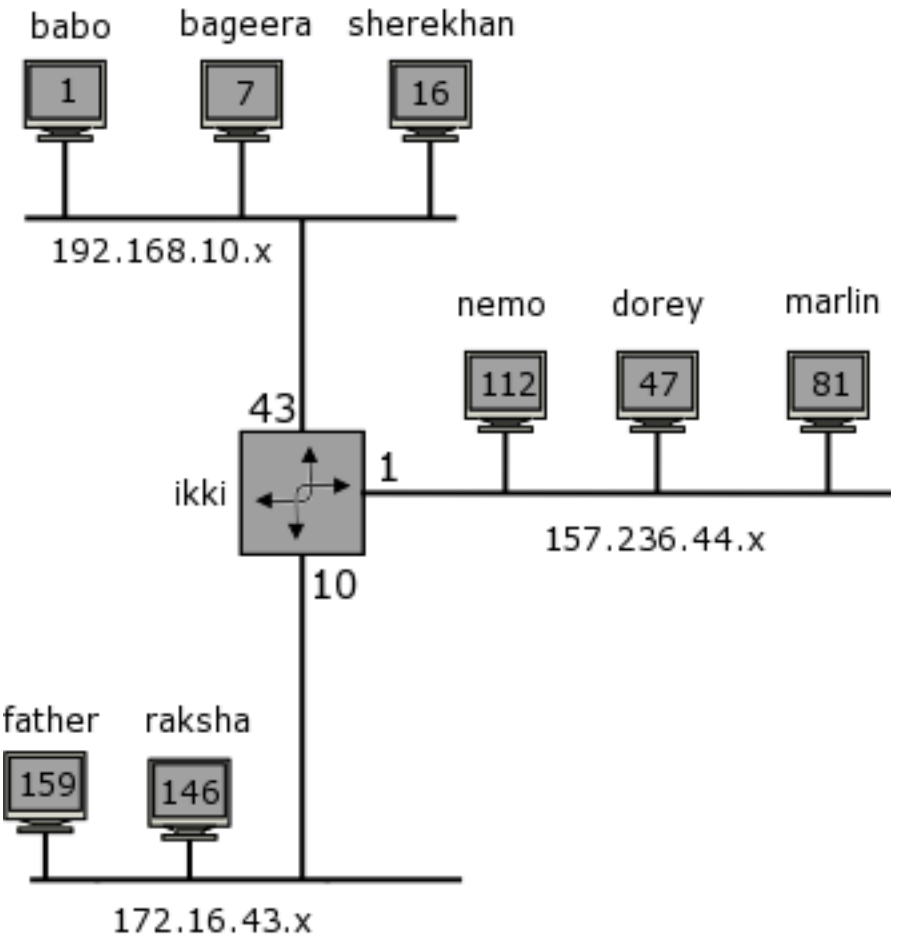
1. The file responsible for controlling BIND usually resides in **`/etc/named.conf`**. Within this file, the administrator can define where the zone files will reside.

2. I like to stick to a standard, and that is to keep all the zone files in /var/named. Of course you needn't follow me - that's entirely up to you!

Configuring BIND can be just that - a bind. So, I will try to highlight areas that will potentially be problematic.

For the purpose of this course, I've decided we'll create a domain called zoo.org.za. For this purpose, I've repeated the network diagram we showed earlier again. Keeping it handy while reading this section will make your life a little easier.

Figure 2.7. The Sample Network



The zoo will contain some of my favorite animated cartoon characters from some of my childhood movies [you will see Nemo, Marlin and Dorey here too - and from that you can draw your own conclusions!-)]

The structure of zone files is essentially the same irrespective of the zone you're configuring. A zone file will contain a set of records that pertain to that zone. These are called resource records.

We will obviously need a zone file for the name to IP address records and another for the IP address to name records. Then again, we need a set of zone files for each new zone we create.

Within our zoo.org.za domain, if we have a felines sub-domain (i.e. Felines.zoo.org.za) we'll need another set of zone files for this sub-domain. But I'm running ahead.

To begin, we'll create the forward zone files (name to IP address). we'll also do everything the 'long-winded' way now, and show you abbreviations at the end.

Resource record begin in column 1 always - no spaces, tabs or other funny stuff or your zone file will not work.

This is for zoo.org.za.zone :

```
zoo.org.za    IN    SOA    baloo.zoo.org.za.  \
                                mowgli.zoo.org.za. (
                                2004022201 ; serial number
                                3h ; refresh
                                1h ; retry
                                1w; expire
                                1h ; minimum
                                )
```

The line begins with the domain we are responsible for - the start of our authority.

Notice how I have specified it in lower case. Case to DNS is unimportant, although it is preserved.[3]

This is an INternet (IN) record type. There are other record types, but for all practical purposes, you will probably only ever deal with Internet record types.

Since we are going to be the authority for this domain, this record is really telling all other name servers that this is our Start of Authority (SOA).

Next we specify the primary master name server for this SOA. In this case, baloo is the primary master name server for the zoo.org.za domain.

You will notice that I have written the name as baloo.zoo.org.za. - a full stop after the za. I'll explain why shortly. ??????<xref to page 18>???????

Then we specify the system contact for this domain. In place of the @ sign, we put a full stop, since @ in the zone file has special significance. So, in this example, mowgli@zoo.org.za is the contact for this zone. If you're having a problem with our zone, contact mowgli.

From here on, everything in the round brackets is merely information for the

[3]My domain for example is QEDux.co.za. I always write the QED in uppercase, and it is preserved in name lookups.

secondary name servers (the slaves). we'll explain more on the meaning of these when we set up the slave server.

The next entries relate to describing the name servers for this zone. Notice again that I include a full stop on the end of the FQDN for the two name servers.

You could of course have 3 or more name servers too, but let's not get too fancy.

```
zoo.org.za IN NS baloo.zoo.org.za.
zoo.org.za IN NS ikki.zoo.org.za.
```

Again these resource records are for the zoo.org.za domain. They are Internet records (IN) and they are the name servers (NS) for this domain.

Finally, we list the two name servers as baloo and ikki.

That done, we can now start describing the resource records for all the other hosts in our domain.

```
bagheera.zoo.org.za.          A 192.168.10.7
sherekahn.zoo.org.za.          A 192.168.10.16
baloo.zoo.org.za.          A 192.168.10.1
ikki.zoo.org.za.          A 192.168.10.43
```

Now, since this is the zone file for this domain, even having other IP addresses in this domain doesn't affect where we put these resource records.

```
tweetie.zoo.org.za.          A 172.16.43.159
raksha.zoo.org.za.          A 172.16.43.146
ikki.zoo.org.za.          A 172.16.43.10

nemo.zoo.org.za.          A 157.236.144.29
dude.zoo.org.za.          A 157.236.144.101
marlin.zoo.org.za.          A 157.236.144.93
ikki.zoo.org.za.          A 157.236.144.1
```

Having defined our zoo.org.za zone file, we should be set to start our name server but remember that I said DNS does both forward lookups (name to IP address) AND reverse lookups (IP address to name). Since these are described in different 'trees' (see Figure 2.4 [25]), we still have not described the reverse zone file.

I like calling my reverse zone files as follows. If I'm describing the range 192.168.10.x then the file is called:

```
192.168.10.zone
```

In this file, we need to define a SOA record as before, with one slight modification.

Whereas previously we were describing the zone zoo.org.za, here we are defining the zone 10.168.192.in-addr.arpa. So my SOA for this zone file reads as follows:

```
10.168.192.in-addr.arpa IN SOA  baloo.zoo.org.za. \
                                mowgli.zoo.org.za. (
                                2004022201 ; serial
                                3h         ; refresh
                                1h         ; retry (1 hour)
                                1w         ; expire (1 week)
                                1h         ; minimum (1 hour)
                                )
```

We need to define the name server records as well as individual resource records for each of the hosts in the forward mapping zone file. I've listed my entries below:

```
; This entry describes the reverse mappings \
                in the 192.168.10.zone file
10.168.192.in-addr.arpa. IN NS baloo.zoo.org.za.
10.168.192.in-addr.arpa. IN NS ikki.zoo.org.za.

1.10.168.192.in-addr.arpa. IN PTR baloo.zoo.org.za.
7.10.168.192.in-addr.arpa. IN PTR bagheera.zoo.org.za.
16.10.168.192.in-addr.arpa. IN PTR sherekahn.zoo.org.za.
43.10.168.192.in-addr.arpa. IN PTR ikki.zoo.org.za.
```

What is different about entries in this zone file from those in the forward zone file described above is the fact that these are pointer (PTR) records instead of address (A) records as before.

We can create similar files for the 157.236.144 network and the 172.16.43 network. I'll leave those to you as an exercise.

Now that we have all our zone files we're ready to begin serving name requests.

Well, not quite . . .

We need a couple of additional zone files. By default, every host on the Internet needs an address of 127.0.0.1. This is the interface referred to as the loopback.

To look at the loopback on your host using the command:

```
ifconfig lo
```

The loopback is used by applications on the local machine to talk to other applications locally. In order for this to continue to work uninterrupted, you will need to include two additional zone files. They are generally included as part of the BIND install, but if not, copy them from the files named.local and 127.0.0.zone as part of this course.

The last zone file is the root zone file. (boy this DNS configuration is really long-winded!).

We may need an updated root zone file as the one shipped with BIND may be a little outdated. It's your responsibility as the DNS administrator to keep this file up-to-date.

In the default install of BIND it should be called something like named.ca or root.ca, which should be sufficient to get us going.

The named configuration file

Now that all our zone files are configured and ready, we need to modify the main named configuration file - named.conf. This usually resides in the /etc directory.

There are many options we can include in this file, but for now, we'll restrict ourselves to those options that will ensure we get a name server running ASAP.

The options directive specifies global options for the name server. Entries will include parameters such as the directory containing your zone files, specification of the forwarder, whether to forward first or forward only, etc.

Then, for every zone we are responsible for, we need a zone entry, and zone entries take the form:

```
zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
    type master;
    file path_name;}
```

Of course, there are many more options that comprise the zone statement, but I've deliberately left them out as I don't want to complicate life too much right now.

Translating this syntax into a practical example:

```
zone "zoo.org.za" IN {
    type master;
    file "zoo.org.za.zone";};
```

Remember to finish each directive and parameter with a semi-colon (;). If you don't you'll experience problems that are often difficult to detect.

The rest of the zones are similar in fashion to the one show above, but I'll include them here to illustrate the reverse mapping named.conf entries.

```
zone "10.168.192.in-addr.arpa" IN {
    type master;
    file "192.168.10.zone"; };
```

Now you will need to add the extra zone entries for the other networks on your pretend network. Remember to include entries for the loopback zone and the root zone as I have included them below.

```
Zone "." IN {
    type master;
    file "named.root"; };

zone "localhost" IN {
    type master;
    file "named.local"; };

zone "0.0.127.in-addr.arpa" IN {
    type master;
    file "127.0.0.zone"; };
```

A point worth noting here is that the root zone, is represented as a full stop (.), since the real representation of "" would look a little confusing.

Error messages

I would really suggest that you keep an eye on your syslog or messages file in /var/log, as this will spit out the errors when or if they occur.

You can even run bind in the foreground so that errors are sent to the console instead of to a log file.

```
/usr/sbin/named -g -d 1 -c /etc/bind/named.conf
```

This will start named in the foreground, spitting all error/log messages to the console.

Starting the DNS Server

Okay folks. we're ready to start our DNS server now and starting named is quite simple.

```
/usr/sbin/named -g -d 1 -c /etc/bind/named.conf
```

This will start named in the foreground, spitting all error/log messages to the console.

Now our server is read to answer queries about our domain, so let's ask some using dig.

```
dig @my-DNS-server-IP-Address ikki.zoo.org.za A

dig @my-DNS-server-IP-Address nemo.zoo.org.za A

dig @my-DNS-server-IP-Address -x 157.236.144.81

dig @my-DNS-server-IP-Address panther.zoo.org.za
```

Your DNS should be capable of resolving these without any problems.

Troubleshooting zone and configuration files

Humph. It did not work, so where to from now?

There are a number of BIND tools that you can use, some of which I'll review later.

However, right now you have a problem. Your DNS is not starting up or, if it is, it's not answering the questions properly. BIND comes bundled with a couple of useful utilities.

Most notably the named-check{zone,conf} syntax checkers which, will by no means pick up every problem in your named.conf file or your zone files, but they will assist

you in isolating the really obvious ones.

So, let's run named-checkconf on our /etc/bind/named.conf file:

```
/usr/sbin/named-checkconf -t /etc/bind
```

By default, this utility will check the named.conf file. It will spew out errors if you have any, otherwise it will be completely silent and just return you to your prompt.

Then we need to check our zone files using the named-checkzone utility.

```
/usr/sbin/named-checkzone  -d \
                    10.168.192.in-addr.arpa 192.168.10.zone
```

which, should return:

```
loading "10.168.192.in-addr.arpa." \
                    from "192.168.10.zone" class "IN"
zone 10.168.192.in-addr.arpa/IN: loaded serial 2004022301
OK
```

If there are problems in your zone files, this should catch the majority of them.

Configuring your resolver (revisited)

You will have noticed that each time we do a lookup on a name we need to type the entire name.

Now we don't necessarily want to do this each time, as it expends valuable carpel energy!!!! For a simpler lookup, we can modify our /etc/resolv.conf file to accommodate the domain. The domain option in the resolv.conf file is not absolutely necessary, since the nslookup and the dig commands will use the hostname, remove the domain part and use this to look up the FQDN.

Thus:

```
nslookup ikki
```

would look up:

```
ikki.zoo.org.za
```

When using dig, the syntax is slightly different:

```
dig +search ikki
```

The search option will use the searchlist or the domain directive in the resolv.conf file, in a similar way to nslookup.

How do we get this functionality?

The resolv.conf file has a number of directives that may be used to speed up or change the way lookup answers are returned.

The nameserver directive

This directive defines the name server that will be used to do DNS queries. There can be up to MAXNS records in this file.

MAXNS is a constant that is set in the C header include file for the resolver. To see how many MAXNS is on your host, look in the resolv.h header file. This file resides in /usr/include on my system, but your mileage may vary depending on your version of Linux. MAXNS is set to 3 for me.

The reason for having more than a single entry is for some degree of redundancy/load sharing.

In addition, an option "rotate" may be used which will rotate the use of the nameserver directive should there be more than one entry.

The search directive

This directive allows the resolver to search through the list of defined domains in order to do a name lookup. An entry may look as follows:

```
search zoo.org.za co.za com
```

```
```

Thus:

```
dig +search ikki
```

will try name lookups as follows:

```
ikki.zoo.org.za
ikki.co.za
ikki.com
```

The first to match will be returned. Note though, that this can be slow and network resource hungry, especially if the name server is remote to your client. So, if you are dialing up at 9600 baud, don't set this option [believe it or not, we here in South Africa, still have some people dialing up at this archaic speed!]

The domain directive

The domain directive and the search directive are mutually exclusive. The domain directive will be used by appending it to the hostname specified on the command line.

Typing:

```
dig +search ikki
```

with the domain directive set to QEDux.co.za, will always look up the name:

```
ikki.QEDux.co.za
```

This is less resource intensive than the search command and you are ill advised to use both of these directives in your file. Read resolv.conf(5) for details on these directives.

The sortlist directive

In the case of our DNS records, ikki.zoo.org.za has 3 different entries in the DNS, since it is multihomed. Which one should be returned?

Suppose for a minute that I am on the 157.236.144 network. Clearly, I don't want the 192.168.10 ikki address to be returned at the top of the list when doing a query.

Similarly, I would be unwise to have the 172.16.43 address returned at the top of the returned addresses. For this reason, the sortlist directive will sort returned answers according to a specified set of criteria.

My sortlist directive is set as follows:

```
sortlist 157.236.144/255.255.255.0 172.16.43/255.255.255.0
```

This should solve the problem. Since each of these directives can be set on a per host basis, they needn't be the same on every host.

In sum then, my resolv.conf file will look as follows:

```
; just to complicate matters, there is NO full stop on the end
; of the domain directive. Semicolons on this file are taken to
;  be comments.
    domain zoo.org.za
    sortlist 157.236.144/255.255.255.0 172.16.43/255.255.255.0
    nameserver 192.168.10.43
    nameserver 196.7.138.45
    nameserver 66.8.48.243
    options rotate
```

Master server shortcuts

Now that we've configured and run our master server successfully, it's time to create some shortcuts.

The zone directive in the named.conf file specifies the domain for this zone. In our examples, we had zone statements as follows:

```
zone "zoo.org.za" IN {
        type master;
        file "zoo.org.za.zone";
};
or
zone "144.236.157.in-addr.arpa" IN {
        type master;
```

```
        file "157.236.144.zone";
};
```

Within our zone files, we had entries as follows:

```
baloo.zoo.org.za. IN A 192.168.10.144
bagheera.zoo.org.za. IN A 192.168.10.7
sherekahn.zoo.org.za. IN A 192.168.10.16
```

Earlier, we stressed the importance of not leaving off the full stop on the end of the FQDN in the zone files. Now we can explain why.

When BIND uses these files to answer a DNS query, it appends the domain name onto the end of the name returned if and only if, there is NO full stop on the end of the name.

Thus, setting up the zone file as follows:

```
baloo.zoo.org.za IN A 192.168.10.144
bagheera.zoo.org.za IN A 192.168.10.7
sherekahn.zoo.org.za IN A 192.168.10.16
```

A query for sherekahn.zoo.org.za will return sherekahn.zoo.org.za.zoo.org.za, since the trailing full stop has been omitted. Not quite what we had in mind!

Now, we can modify our zone files as follows (for the zone zoo.org.za):

```
baloo         IN A 192.168.10.144
bagheera IN A 192.168.10.7
sherekahn IN A 192.168.10.16
bear          IN CNAME baloo
panther IN CNAME bagheera
```

or our reverse zone files as follows (for the zone 144.236.157.in-addr.arpa):

```
112 IN PTR nemo.zoo.org.za.
92 IN PTR dude.zoo.org.za.
81 IN PTR marlin.zoo.org.za.
```

Since the domain is specified in the zone directive of the named.conf file, we are able to omit the full specification. Note here that if we wish to specify resource records in this format, we will need to omit the domain as well as the ending full stop.

The $ORIGIN directive

The most useful of the directives is the $ORIGIN directive.

This directive can be placed at the beginning of a section in the zone files, which shortens both the amount of typing you need to do to set up the DNS and makes it a lot easier to read.

The $ORIGIN persists until another $ORIGIN directive is encountered.

By default, in the absence of a $ORIGIN directive, the zone directive in the named.conf file specifies the domain name.

An example shown below illustrates the use of $ORIGIN:

```
$ORIGIN zoo.org.za.
Bagheera A 192.168.10.7
baloo         A 192.168.10.1
bear         CNAME baloo
or
$ORIGIN 43.16.172.in-addr.arpa.
10         PTR ikki.zoo.org.za.
146         PTR raksha.zoo.org.za.
```

This could be used when we create sub-domains.

Initially, we could start with a domain zoo.org.za, and within this zone file, change the ORIGIN to felines.zoo.org.za

The @ directive

Another shortcut is using the @ instead of the domain name.

If the domain name is the same as that specified in the zone directive, then simply using the @ will be equivalent to the domain.

If the zone is "43.16.172.in-addr.arpa", then we could represent this in the zone file as:

```
@    IN    SOA    baloo.jungle.org.za. mowgli.jungle.org.za.\
```

```
                (
                2004022406 ; serial
                10800         ; refresh (3 hours)
                3600          ; retry (1 hour)
                604800 ; expire (1 week)
                3600          ; minimum (1 hour)
                )
                NS          ikki.zoo.org.za.
                NS          baloo.zoo.org.za.
```

where the @ is used to refer to the zone.

Notice too that for the NS resource records, there is no domain specified. This is because the domain is inherited from the previous entry (which in this case was the @, which in turn was "43.16.172.in-addr.arpa").

The NS resource records could have been written as:

```
43.16.172.in-addr.arpa NS ikki.zoo.org.za.
43.16.172.in-addr.arpa NS baloo.zoo.org.za.
```

Or alternatively as:

```
@                NS ikki.zoo.org.za.
@                NS baloo.zoo.org.za.
```

The $TTL directive

The time to live is the amount of time a name server can cache a record. Cached for too long and the data may become stale. Cached for too short a time and the zone transfers will consume network bandwidth. Clearly we need to set an optimum time for an entry to remain in the cache. Remember too that both positive and negative replies will be cached, but both need a time to live.

There are a couple of places to set the time to live. In the SOA resource record, the last entry specifies the TTL. This is the time to keep NEGATIVE responses to a query.

Then there's the $TTL variable, which can be set at the top of the zone file.

```
$TTL 3h
```

would allow records (both positive and negative answers) on the slaves to be cached for a maximum of 3 hours.

Finally, each resource record entry can contain a time to live. This will for only that entry to the timed out sooner or later. One of the uses of this is when you know a host will be changing networks. Thus, 3 hours may be a little long before updating the slave servers.

One could easily add an entry as follows:

```
dude  1h IN  A    157.236.144.92
; explicit make the TTL of this RR 1 hour
```

Configuring the Slave Name Server

The slave name server

we've spent a great deal of time configuring the master server. Now that it's up and running, let's turn our attention to the slave name server(s). In a network, we can only have a single master server, but we can have multiple slaves.

The larger ISP's will, on the whole, have many slaves to ease the load across the network, which could potentially be a bottleneck.

As mentioned previously, the slaves are in no way less important than the masters, and slaves in their own right can be masters of a different zone. In this way, one master can be the slave of a master from another zone.

The primary difference between the master and the slave however is the fact that slaves get their information from masters by the process of zone transfers. we'll see zone transfers occur as soon as our slave is going.

Slaves are really easy to configure, since they don't need any zone files. Well, not quite. They do require zone files for the localhost and 127.0.0 zones. Since these files are standard across all name servers, and since it makes little sense to transfer this zone between one name server and the next, they should be present on the slave. Again, they will probably reside in /var/named (on Debian they are in /etc/bind/).

We will also require an **`/etc/named.conf`** (on Debian again that's in /etc/bind/named.conf).

For the other zones, the slave will transfer them from the master automatically.

The primary differences in the layout of the named.conf files is as follows:

Slave name servers do not have:

```
type master;
```

in their zone entries but rather

```
type slave;
```

Slave servers specify a slightly different name for the zone files they draw from the master, although this is not a prerequisite.

```
file "bak.172.16.43.zone";
```

or

```
file "zoo.org.za.backup.zone";
```

A masters keyword indicate who the master server is for this zone:

```
masters { 157.236.144.1; };
```

So a zone entry for the zoo.org.za zone would look as follows:

```
zone "zoo.org.za" {
        type slave;
        file "zoo.org.za.backup.zone";
        master { 157.236.144.1; };
}
```

Now we're ready to start our slave server. We start it again just like we started the master server. If you start it in the foreground as before:

```
named -g -u named -c /etc/named.conf
```

You should see the IXFR (inter-zone transfer) taking place:

```
Feb 24 14:27:59.117 \
            zone zoo.org.za/IN: transfered
            serial 2004022301
Feb 24 14:27:59.117 \
            transfer of 'zoo.org.za/IN' from
            192.168.10.144#53: end of transfer
Feb 24 14:27:59.118 \
            zone zoo.org.za/IN: sending notifies
            (serial 2004022301)
Feb 24 14:27:59.631 \
            zone 10.168.192.in-addr.arpa/IN:
            transfered serial 2004022301
Feb 24 14:27:59.632 \
            transfer of '10.168.192.in-addr.arpa/IN'
            from 192.168.10.144#53: end of transfer
Feb 24 14:27:59.634 \
            zone 144.236.157.in-addr.arpa/IN: transfered
            serial 2004022301
Feb 24 14:27:59.634 \
            transfer of '144.236.157.in-addr.arpa/IN'
            from 192.168.10.144#53: end of transfer
Feb 24 14:27:59.634 \
            zone 10.168.192.in-addr.arpa/IN: sending notifies
            (serial 2004022301)
Feb 24 14:27:59.635 zone \
            144.236.157.in-addr.arpa/IN: sending notifies
            (serial 2004022301)
Feb 24 14:27:59.639 \
            zone 43.16.172.in-addr.arpa/IN: transfered
            serial 2004022301
Feb 24 14:27:59.639 \
            transfer of '43.16.172.in-addr.arpa/IN'
            from 192.168.10.144#53: end of transfer
Feb 24 14:27:59.639 \
            zone 43.16.172.in-addr.arpa/IN: sending notifies
            (serial 2004022301)
```

Once this is done, you can change your nameserver directive in the resolv.conf file and test it. Alternately, you can issue the command:

```
dig @my-slave-DNS-name-server +search ikki
```

which should be able to answer the query without any problems.

Slave server settings in the SOA resource

record

Now that we're dealing with slave servers, let's return to consider the numbers in the SOA record.

```
zoo.org.za. IN SOA baloo.zoo.org.za. mowgli.zoo.org.za. (
                        2004022401; Serial
                        3h; Refresh after 3 hours
                        1h; Retry after 1 hour
                        1w; Expire after 1 week
                        1h; Negative caching TTL of 1 day
                        )
```

Serial and refresh

Serial is a number indicating the 'revision' of the zone file. Whenever a change is made to the zone file, this serial number should be updated. It is only by updating the serial number, that slave servers will themselves be updated. Each "refresh" time, the slaves will contact the master server, and check whether the serial number has changed. If so, the slave will do an interzone transfer of the masters zone files.

The DNS administrator has the responsibility of updating the serial number for each change made to the zone file. Any number may be used, being incremented in sequence for every change.

It is common practice however to use a serial number in the form of:

```
YYYYMMDD##
```

where

```
YYYY --- is the year, including the centuary,
MM   --- is the month,
DD   --- is the day the zone file is updated and
##   --- is the nth updating of the zone file that day.
```

do not be tempted to format the date in any other style e.g. DDMMYYYY, since this will not produce sequential numbers, and your slaves will give unpredictable results

Expire time

The slaves cannot keep answering DNS requests indefinitely. In fact, if a slave has not received an update from the master for more than a week, the slave will not answer any further questions.

So, why do the master zone files contain the expiry time?

Since slaves obtain their zone files from the master, this in effect tells the slaves how long to answer queries in the absence of the master zone transfers.

A point worth noting here is that the slaves store the zone files locally just in case the master is not available when the slave starts up. In this case, the slave will read the zone files locally rather than forcing a zone transfer.

More on this later when we discuss rndc the section called “Rndc” [57], the DNS name server control software.

Retry time

This it the time that the slaves will wait before retrying to get the zone files from the master if the master was down the first time they tried. In fact the retry time is the time between one retry and the next retry.

TTL time and negative caching

This is the time that the NEGATIVE cached entries will remain in the slave servers.

(See the explanation earlierthe section called “The $TTL directive” [49] for other TTL directives.)

Mail exchange record (MX)

On the Internet, one of the most important jobs is that of sending email.

The only problem is that one mail server needing to send email to another, needs to know what the mail servers IP address is. No problem you say, DNS will be able to solve that.

The problem is that a domain might have 10 hosts in the DNS. Perhaps more. Given this, which host is the SMTP server?

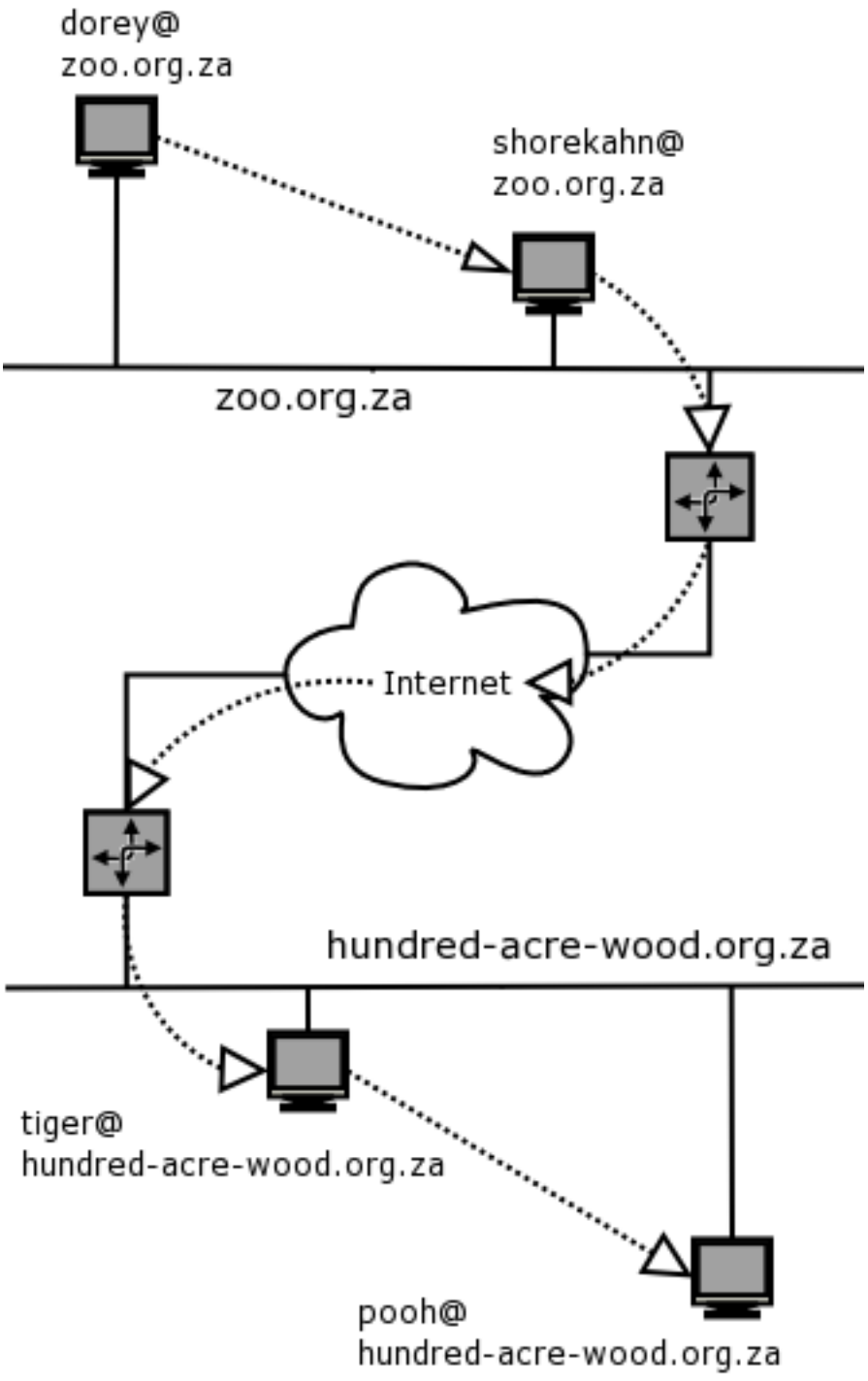
Now the sending host could test each server in the domain, to determine which of the 10 hosts is SMTP capable, but apart from the inordinate amount of time this would take, and the unnecessary network traffic it would generate, this is certainly not a solution the DNS designers would want to consider.

To solve this problem, there is the MX or mail exchanger record (just another type of resource record actually).

Generally a domain will have at least one MX record, but may in fact have any number to solve issues of redundancy. The MX record specifies the host(s) responsible for transfer of email for that domain. So, when a sending mail server wishes to send email to a user at a domain, it would request the MX record for the receiving domain, and thus be able to initiate a SMTP connection to the host.

Figure 2.8 [55] illustrates the process of two SMTP servers exchanging email.

Figure 2.8. SMTP Process



humphrey@zoo.org.za (humphrey is a user on the host dorey) wishes to send an

email to pooh@hundred-acre-wood.org.za

1. The host, dorey, forwards the email to the SMTP MTA (sherekahn.zoo.org.za) - this is the mail exchanger for the zoo.org.za domain.
2. Sherekahn.zoo.org.za requests the MX record for the domain hundred-acre-wood.org.za
3. The DNS query returns the MX record for hundred-acre-wood.org.za which points to the host tigger.
4. Sherekahn then contacts tigger and they begin the process of sending the email message.
5. At the end of the conversation, they both 'hang up', and since the mail has been successfully received by tigger, the email is deleted from sherekahn.zoo.org.za.

Since a domain can have more than a single MX record, each MX record will have a 'preference' value. This solves problems of routing loops, and also allows a single server to bear the brunt of the mail load. Should this server fail, alternate servers (admittedly with higher preference values) will take over the responsibility.

The numbers are simply an ordering scheme. When mail is delivered, the MX record with the lowest preference value will be the place where mail is delivered. Should the mailer fail to deliver the message to this MTA, then the next highest preference mail server will be selected.

Finally, two or more MX record can contain the same preference value. In this case, which mail server receives the mail is irrelevant.

Rndc

rndc is a utility that allows us to manipulate our BIND server.

It is useful in that it allows us to stop and restart the server, force refreshes of the zone files or slave zone transfers, flush server caches, etc.

The most obvious thing you want to do with rndc is to refresh the zone files on the slave.

To test this, you will need to make a change to the master zone files. In fact, simply changing the serial number in the zone file is enough.

Once you've changes the serial number, issue the rndc command:

```
rndc reload 144.236.157.in-addr.arpa
```

Then look at your log file (usually /var/log/messages or /var/log/syslog), and you should see the altered zone file is reloaded. You should be able to tell using the serial number.

```
Feb 24 14:51:34 \
            baloo named[559]: zone \
            144.236.157.in-addr.arpa/IN:
         loaded serial 2004022302
Feb 24 14:51:34 \
            baloo named[559]: zone \
            144.236.157.in-addr.arpa/IN:
        sending notifies (serial 2004022302)
```

Now, on the slave, issue the command:

```
rndc refresh 144.236.157.in-addr.arpa
```

and the zone files will be transferred to the slave.

Other rndc commands will produce statistics on the servers, reconfigure the name servers and more.

A caching only name server

Sometimes you may not wish to set up a full name server.

Typically, users that are on the wrong end on a dial-up Internet connection (the slow end), might wish to configure a caching only name server. In this way, each time a query is done to the DNS server, it will be cached locally, speeding up future lookups to the same address.

Caching only name servers don't have all the advanced features we've been talking about in this module - they are only able to cache and answer queries that they already know about.

Configuring a caching only name server is also a whole lot easier to set up and administer than it's older cousin a master/slave name server.

In order to configure a caching only server, one needs to configure the

`/etc/named.conf` file.

Instead of the zone entries, include the following options:

```
forwarders { 192.168.10.144; 196.14.187.146; };
forward  first;
```

This will ensure that all queries are forwarded to the name servers listed in the forwarders directive, while the "forward first" will first try to forward the query before trying to find the answer itself. that's it.

Now start the name server and begin building a cache of all the sites visited.

I have deliberately left out some sections in this module - those being DNS security, round robin DNS, and configuration of sub-domains. I hope to add these chapters in future releases of this course. If this specifically interests you, feel free to get the excellent DNS and BIND book referred to in resources below.

Research Resources:

DNS and BIND (4th Edition) by Paul Albitz and Cricket Liu, published by O'Reilly Press. ISBN: 0-596-00158-4

This page intentionally left blank

Chapter 3. The SQUID proxy server

What is a proxy server?

In the early days of the development of the Internet, users were allowed unlimited access to it's resources.

Since many companies had few Internet users, this presented very little problem. However, the Internet has grown, probably exceeding what the original designers ever dreamed and with this expansion, comes the need to increase the speed.

One method of achieving this is by keeping copies of the web sites visited by people within an organization, so that, should another individual visit the same site within a short time span from the original visit, the second visitor would get significantly improved response as the web page would have been stored (cached) on a local server. This not only reduces latency, but also has the added advantage of limiting the consumption of expensive bandwidth by the users themselves. This method of increasing the speed is done by setting up a proxy server.

There are some additional advantages to using a proxy server, and these are:

1. Since all users are now forced to use the proxy instead of going directly to the Internet, a degree of control is offered to the system and network administrators.

2. Using the proxy they have the ability to restrict the users to particular times of the day, or by machine name, or by network IP address range.

3. Added to this, they have the ability to block the users from visiting undesirable sites, do usage accounting and force authentication prior to accessing the Internet.

All this adds up to a very useful tool and under Linux, this tool is called SQUID.

So how does a proxy work?

Well, in much the same way a proxy for voting in your next Chairman would work. Assuming you are eligible to vote, but on this particular day (that the elections are being held), you happen to be out of town. However, having anticipated this, you ask whether you can cast your vote by proxy. So, finding a person whom you trust, you secretly tell them your vote and they, on your behalf, vote for you in absentia.

The proxy server is just like this. Instead of your Web browser heading for the Internet when you type in an URL, the proxy intercepts the request and asks for the page on your behalf.

The reply comes back to you, but via the proxy and the proxy server stores the reply, just in case you (or anybody else in your organization asks for it again shortly).

Thus, if the proxy were doing this for every individual within the organization, it would quickly build up a cache of all these requests, serving the same pages in a much reduced time on future requests.

SQUID configuration

Squid is governed by a configuration file, normally residing in **`squid.conf`**

There is little other documentation shipped with squid in the form of manual or Texinfo (used by the info command) pages.

However, in addition to the well-commented configuration file, FAQ's and documentation also reside in the **`/usr/share/doc/squid`** directory. The official web site is not www.squid.org as may be expected, but http://www.squid-cache.org.

`squid.conf` is a mile long and has many options (129 in all I think). Luckily though, in order to get SQUID going as quickly as possible (although far from optimal), only 2 settings need be changed.

We will look at these two changes in this course but we will also spend some time examining some of the other 127 configuration options.

Setting the port on which SQUID listens

Out the box, SQUID will listen on port 3128 (the default). While it may be convenient to listen on this port, network administrators often configure the proxy to listen on port 8080.

This is a non-well-known port (ports below 1024 are well-known ports and are restricted from being used by ordinary users processes), and is therefore not going to be in conflict with other ports such as 80, 443, 22, 23, etc. SQUID needn't be restricted to one port. It could easily be started in 2 or more ports.

In the example below, I will start SQUID on both ports (8080 and 3128).

```
http_port 8080 3128
```

Additionally, if you have multiple network cards in your proxy server, you may wish to restrict the proxy to start on port 8080 on the first network card and port 3128 on the second network card.

```
http_port 192.168.0.1:8080 172.16.43.1:3128
```

One point worth noting is that there should be no other services running on these ports prior to starting the proxy server. If there are, SQUID will not start and you will need to consult the log files to determine what has caused SQUID this indigestion.

Finally, if you don't like the ports I've selected here, you are most welcome to choose your own, provided you adhere to the rule in the paragraph above.

If you choose a port below 1024, SQUID will need to be run as the root user, but this is not really a problem, as a new process will be spawned and this new process will run as the squid user.

The SQUID user

SQUID should not be run as root. In it's default mode, if SQUID is started as root, it will run as the effective user 'nobody', and the effective group 'nobody'.

As mentioned earlier, if you wish to run your squid on a port below 1024, you will need to start is as the 'root' user.

There are another two configuration options in the **`squid.conf`** file indicating which user and group should be used for the server.

cache_effective_group and cache_effective_user are the configuration files settings that are needed.

It is advised that you create a user squid (if it's not already there) as well as a group 'squid' and change the configuration file accordingly.

Once the user and group that you are going to use has been established we need to set up access to our proxy.

Access control to the proxy server

Access control is determined using two sets of configuration parameters. The first is the access control list (acl), and the second is the http_access list (http_access). The 'acl' really defines a synonym that we can use within the http_access list.

The general form of these acl lists is:

```
acl aclname acltype string1 ... | "file"
```

'acltype's	can be any of (this is not the whole list!):
src	the source address of the packet (e.g. Your IP address)
dst	the destination address of the packet (e.g. The IP address/domain to which this packet was destined)
srcdomain	the source domain of the packet (for this, SQUID must do a reverse name lookup to determine the source domain, adding to the latency)
dstdomain	as in the source domain, only the domain for which the packet is destined
srcdom_regex	a regular expression describing the source domain (e.g. QEDux.*)
dstdom_regex	as in srcdom_regex A regular expression using the source or the destination domain. Be aware that this can cause delays as a result of lookups.
Time	specify the time at which hosts can connect e.g. acl allowed_clients src 192.168.0.0/255.255.255.0 acl regular_days time MTWHF 10:00-16:00 http_access allow allowed_clients regular_days

Once we've defined the acl, we can use that to define the http_access files. We showed an example in the time acltype above, but let's look at others.

The general format of the http_access directive is:

```
http_access allow|deny [!]aclname ...
```

Thus, if we have an acl as follows:

```
```

```
acl allowed_clients src 192.168.1.0/255.255.255.0
```

The http_access line could read:

```
http_access allow allowed_clients
```

A couple of points to note regarding http_access lines are:

Each line is 'OR'ed with the next. Thus, having two lines as follows:

```
http_access allow allowed_clients
```

OR

```
http_access deny !allowed_clients
```

and this would allow the allowed_clients, but deny the clients that don't conform to the allowed_clients source address. The first match will cause squid to accept the request and no further lines will be matched. In typical OR style, "TRUE or anything" will always yield a TRUE result.

So order is important in the http_access lines.

If you consult your **`squid.conf`** file, you will notice in the acl there is an entry as follows:

```
acl all src 0.0.0.0/0.0.0.0
```

with an associated line, at the end of all http_access lines, as follows:

```
http_access deny all
```

The result is to deny all clients that have not matched a previous http_access rule. A 'catch-all' as it were.

Furthermore, each entry on the http_access line is 'AND'ed with the next entry.

In the 'time' example mentioned above:

```
http_access allow allowed_clients regular_days
```

this would be interpreted as:

```
http_access allow allowed_clients AND regular_days
```

This would be stating that if the client is on the 192.168.0.x network AND the time is between 10am and 4pm, allow access through the proxy.

Obviously we would also need to add a deny line somewhere below this, otherwise even though this line might not match, the client would still be allowed through the proxy.

let's look at some examples of acl's (comments above the lines indicate what I'm trying to achieve using them):

```
# Set the source address for the 'alias' QEDux.net.work
# (clients on this network in range 10.0.1.x)
        acl  QEDux.net.work src 10.0.1.0/255.255.255.0

# Set the source address for the 'alias' QEDux.net.home
# (clients on this network in range 192.168.43.x)
        acl  QEDux.net.home src 192.168.43.0/255.255.255.0

# Although defender is on the network 10.0.1.x,
# I am specifically specifying it based on it's MAC address
        acl  defender_arp         arp 00:01:03:8C:FB:01

# Disallow access to any url containing the word \
                     sex or porn or adult.
# Note too the use of the -i option, which will \
                     match the RE in a
# case insensitive manner
        acl blacklist_sites         url_regex -i sex porn adult
```

Now for the http_access lines:

```
# Ensure that this is not defender that is trying \
                     to use the proxy
```

```
        http_access deny defender_arp

# Ensure that anybody trying to use the proxy is \
                    not going to an
# adult/porn/sex site
        http_access deny blacklist_sites

# Otherwise allow users on the domains \
                    QEDux.net.{home,work} to
# use the proxy (and thus use the Internet)
        http_access allow QEDux.net.work \
                    QEDux.net.home

# Otherwise deny all clients that don't fit these patterns
        http_access deny all
```

By default, SQUID as shipped in it's shrink wrapping, only allows access to the localhost using the http_access directives. Thus, in it's simplest form, you will be required to add an acl line for your network, as well as an entry in the **`squid.conf`** file to allow access from hosts on your network.

A comment in the **`squid.conf`** file as follows shows where you should add your own rules:

```
.....
#
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS \
                    FROM YOUR CLIENTS
#
.....
```

Some final notes regarding blacklisting sites are:

Instead of having a regex that will match sites, you could include all the blacklist sites in an external file, and change your acl to read as follows:

```
acl blacklist_sites url_regex "/etc/blacklisted_sites.txt"
```

Now all you would need to do is to add the sites you which to blacklist in this text file and bingo, you've solved a problem.

It is worth noting that denying access to certain sites will probably require that you, or your company, has a policy laid down in advance that employees are aware of. Post contractually laying down the law is not legal and you (or your company) could find yourselves in hot water. It is a warning, and with users becoming more aware of

their rights, it is worthwhile consulting a lawyer prior to undertaking such a policy on the proxy.

My personal means of dealing with 'abuse' of the Internet it to automate a process of publishing the users of the Internet in order of sites they've recently visited. That way, if Joe's name appears at the top of the list for www.playboy.com, it could become an embarrassment at tea time when people begin to quiz him about it!

Exercises:

1. Configure a acl/http_access directives that will allow times of usage of the Internet to lunchtime and after 5pm.

2. Configure an acl/http_access directive to allow users on the 172.16.43 network, but deny particular hosts 172.16.43.149 and 153.

3. Configure an acl/http_access directive to disallow users with the MAC addresses 00:10:11:96:A1:C3, C5 and 96:EF:15

4. Given the following http_access directives, what is the outcome: The acl is given as follows:

```
acl Safe_ports port 80 21 443 563 70 210 1025-65535
acl CONNECT method CONNECT

http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports

acl ip_acl src 192.168.2.0/24
acl time_acl time M T W H F 9:00-17:00
http_access allow ip_acl time_acl

acl hosts1 src192.168.0.10
acl hosts2 src 192.168.0.20
acl hosts3 src 192.168.0.30
acl morning time 10:00-13:00
acl lunch time 13:30-14:30
acl evening time 15:00-18:00

http_access allow host1 morning
http_access allow host1 evening
http_access allow host2 lunch
http_access allow host3 evening
http_access deny all
```

Setting the cache_dir

At this point you may be itching to start your proxy, but wait. There's more!

Out the box, SQUID ships with a default cache size of 100Mb and this may need to be changed depending on how large a site you proxy is handling.

If you are just testing at this stage, leave the defaults. However, you should be aware of what the figures mean after the path to the cache directory.

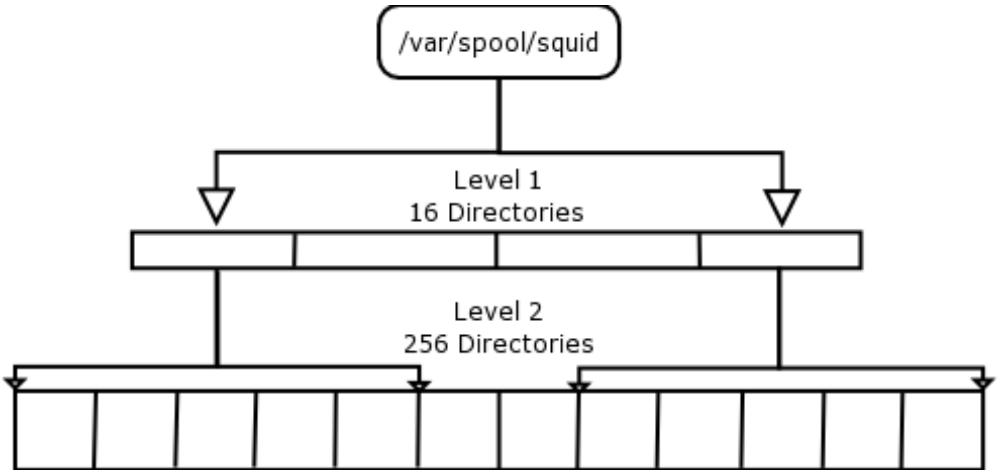
The general format of the cache_dir directive is:

```
cache_dir Type Maxobjsize Directory-Name Mbytes \
                    Level-1 Level2 [..]
```

where :

- Type is generally ufs (although it can be other things too - consult the **`squid.conf`** file)
- Maxobjsize is the maximum object size the storage directory supports. This can be omitted
- Directory-Name is the name of the top-level directory. You may well wish to store your cache in **`/var/cache`** or **`/var/spool/cache`** or **`/var/spool/squid`**. You would specify this here.
- Mbytes, by default out the shrink-wrapping, this is 100Mb. However this is a really small cache and it will most likely be exhausted quite soon when using the cache - even for a small organization. Remember that exhausting the cache will not affect your access to the Internet, but will certainly increase latency to the sites you are visiting.
- The last two entries, Level-1 and Level-2 are the number of first-tier directories that will be created under the Directory-Name, and Level-2, the number of 2nd tier directories under the first-tier

Figure 3.1. Squid Directory Levels



Note that Squid does not automatically create the hierarchy of directories required to cache your pages. You can do this using:

```
/usr/sbin/squid -z
```

The best means of starting squid first time is to start it by hand. Assuming you know where your squid binary lives, you could do something like:

```
/usr/sbin/squid -N -d 1 -D
```

This will not put squid into the background, but will spew all the output to the console.

- The -d 1 option is the amount of debugging information it will show. Increasing the number after the -d will show more debug information.
- The -N option will not put squid into the background, but it will continue to run in the foreground, while the -D option will disable initial DNS tests.

If, on the output, you see:

```
Ready to serve requests
```

your proxy server is ready to be used. If not, read the log carefully to determine what the problem is. Once you have tested the proxy, you can **CTRL-C** this squid process and start it as the init scripts would do (in Debian that would be **/etc/init.d/squid start**).

Testing your proxy

In your class environment, you may or may not have access to the Internet. If you don't, that's no problem. If you do, that's an added bonus. I'm going to assume you do not have access to the Internet. You can simulate an Internet connection using your web or ftp servers, but since you probably don't know how to set these up yet, let's leave that till we've finished this course. Your final lab will require that you configure all these services from scratch on a clean Linux install.

In order to test your proxy, you may choose any web browser. I will cover two here. The first is lynx.

Lynx is a text-based web browser that is fast and light. It is very useful for ignoring those slow-to-download images and the pesky banner ads that make the World Wide Web so slow.

Wait! In order to get lynx to use the proxy, you will need to set at least one environmental variable: http_proxy

```
export http_proxy="http://calamari.jungle.org.za:8080"
```

"http://calamari.jungle.org.za:8080" should be replaced with the domain name or IP address of your machine on which you configured Squid. e.g.. **export http_proxy="http://192.168.0.1:8080"**

Now, start lynx with:

```
lynx www.windowsintoafrica.com
```

Of course, we don't have Internet access, so the www.windowsintoafrica.com will time out, but we can still consult the squid log files to determine whether the client (lynx) used the proxy or not.

Perform a tail on the access.log file (usually in **`/var/log/squid`** - if yours is not there, then look in the **`squid.conf`** file for the cache_access_log directive).

```
tail -20 /var/log/squid/access.log
```

You should see something similar to:

```
59437.023  39923 192.168.1.24 TCP_MISS/504 1139
GET http://www.windowsintoafrica.com/favicon.ico - NONE/- -
```

Here the TCP_MISS means that the proxy was missing this file, and therefore would need to contact the site and retrieve it on your (actually lynx's) behalf.

Lynx error message

Of course, the proxy cannot do that so lynx will return an error such as:

```
ERROR: The requested URL could not be retrieved

                        ERROR

The requested URL could not be retrieved
     ____________________________________________________

   While trying to retrieve the \
                   URL: http://www.windowsintoafrica.com/

   The following error was encountered:
     * Connection Failed

   The system returned:
    (110) Connection timed out

   The remote host or network may be down. \
                   Please try the request again.

   Your cache administrator is hamish@QEDux.co.za.
     ____________________________________________________

   Generated Fri, 20 Feb 2004 08:11:35 GMT \
                   by calamari.jungle.org.za
   (Squid/2.4.STABLE6)
End of error message.
```

If you were using a GUI web browser (say Mozilla or similar), you could configure the browser to use the proxy by following the settings:

```
Tools -> Options -> General -> Connection Settings
-> Manual Proxy Configuration -> HTTP Proxy
```

And enter your proxy (using the DNS name or the IP address) as well as the port on which your proxy is listening.

Now, try to go the same site: www.windowsintoafrica.com, and you should get a similar response to that returned by Lynx.

Whether you have Internet access or not, you should see an entry appear in your access file indicating that your http request was redirected to the proxy.

Automatic client proxy configuration

In the mid '90s, Netscape developed Javascript to address client-side processing (do NOT get Javascript confused with Java, SUN's powerful cross-platform language).

This (at the time) added a huge amount of functionality to web pages, in that some of the page could be 'dynamic'. The browser could understand this JavaScript code and it had the responsibility for processing the code. On the face of it, it looked like an answer to 'static' boring web pages, but of course, came with a whole host of inherent issues. Nonetheless, JavaScript is part of web programming today and I don't think it's going to go away in a hurry.

We can use this functionality in configuring our browser. If visiting 100 client machines fills you with dread, then this may be one way of solving the problem. Of course, you will need to visit your 100 clients at least once, but if anything changes in the future, no need to revisit them. There are of course other ways of skinning this cat (like editing the registry on Windows machines using an automated logon script), but we're certainly not going to delve into that here.

Here I am referring to a proxy.pac script, which is a very simple script, understood by most (if not all) modern browser that will automatically configure the browser for the Internet via your proxy. Of course it does not need to be called proxy.pac, but I use that name here, so let's go with the flow.

In order to ge this right, you will need to:

1. Have a web server set up on your network which will contain the proxy.pac script

2. Configure your browser (as in the GUI configuration above) to use Automatic Proxy configuration URL. In this box, you will need to enter the url,

"http://platapus.jungle.org.za/proxy.pac" where platapus is the name of the web server.

I've included my simple proxy.pac script below, so take a look at it, and then visit http://wp.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html for a list of functions that can be included in your own proxy.pac script.

My proxy.pac script:

```
function FindProxyForURL(url, host) {
        if ( isInNet(host, "192.168.1.0", "255.255.255.0") ||
                dnsDomainIs(host,".jungle.org.za") ||
                dnsDomainIs(host,".QEDux.co.za") ) {
                return "DIRECT";
        }
        else {
                return "PROXY calamari.jungle.org.za:8080";
        }
}
```

This script will set the browser to go directly to sites in the 192.168.1.x network or in the domains QEDux.co.za or jungle.org.za domains, bypassing the need for the proxy. If however, the client machine tries to go to anything else, this script will have configured the browser to use the proxy calamari.jungle.org.za. Bingo, your 100 (or more) browsers are configured. If at a later stage you split your network, putting in a secondary proxy, all that needs to change is the proxy.pac file. No need to ever revisit the individual client workstations.

Setting the cache administrator

We skipped many configuration options earlier in our eagerness to get our proxy working.

let's return an look at a couple more. The first of these is setting the cache administrator. This is the person (probably yourself) responsible for maintaining the proxy.

It would be nice if the user, on obtaining an error on their web browser, could have an email address of someone to contact if they are having problems leaving the local network.

The cache_mgr directive offers just such a solution.

```
cache_mgr hamish@QEDux.co.za
```

```
```

Not only is this email address printed on the bottom of pages when the proxy cannot retrieve the page (as shown above), it is also the person who will receive email in the event that the proxy goes legs-up.

Speeding up the response of your proxy

The process of retrieving a page from the Internet is not slowed by using a proxy except when the requested page cannot be retrieved. In this case, the proxy will try for up to 2 minutes (120 seconds) by default to try to retrieve the page.

This can be annoying to the user, and will certainly have users knocking on your mailbox complaining.

One means of speeding this up is to change the connect_timeout time. If you make this too short of course, the page will never be retrieved since the latency on the Internet may well be significantly slower than 20 seconds (although I would look at my Internet access connection if this was the case!).

I have changed my connect_timeout to 20 seconds and one means of testing this is by issuing the following commands:

```
date +"%s";lynx www.nowhere.biz
```

Compare the time in the access.log file for the www.nowhere.biz entry to that on the command line as output by the date command.

Since squid prints the times in UNIX formatted time (i.e. Number of seconds since 1/1/1970), it will be easy to subtract the time as displayed by 'date' from the time in the access.log file and this should show you the connect_timeout.

Hierarchies of proxies

We have covered the basics of getting your proxy server up and offering better response to your clients.

Proxies can operate in a hierarchical manner. Supposing your ISP has a proxy that stores HITS from all their clients. Clearly, their proxy will be larger than yours since they most probably have more client workstations 'hitting' it on an hourly basis.

What would be nice is to use their proxy as well as yours. Thus, if a web page is not cached on your proxy, your proxy would query the ISP's proxy to see whether it has the page cached in it's tree. If so, it has saved you time and possibly money, as this may be an international site that is cached, and many ISP's in South Africa have differing rates depending on whether the sites you visit are local or international.

The configuration of hierarchical proxies is well outside of the bounds of this course, and of course you would need another proxy on the network in order to see the effect of this, but at least you know the functionality is there and can be configured at some future date.

In a future update on this course, I will include configuration of squid to handle authentication of users.

Exercises:

1. Configure squid to allow access to the internet for your network. Assume your network is in the IP address range 168.158.14.x. You would disallow the workstation 168.158.14.24 access to the Internet as they have been known to be malicious in the past. Deny all FTP access out of your network.

2. Ensure that you squid server has a minimum of 200Mb of space allocated to the cache and ensure that a minimum of 32 directories are created on startup at the tier-1 level. Tier-2 should have the standard 256 directories.

References:

Resources used when compiling the course:

1. SQUID website address: http://www.squid-cache.org

2. An excellent squid tutorial written by Oskar Pearson.
http://squid-docs.sourceforge.net/

3. DS Oberoi's article on setting up a squid cache server:
http://www.linuxfocus.org/English/March2002/article235.shtml

4. Henrik Nordstrom's Squid workhttp://squid.sourceforge.net/hno/

5. Squid ACL Proxy Authentication with External Programs
http://www.devet.org/squid/proxy_auth/

Chapter 4. Configuration of Exim or Sendmail Mail Transfer Agents

Introduction:

The default mail transfer agent (MTA) which Debian comes installed with is called Exim.

A MTA is a system which will receive, spool, store and deliver e-mail for MUAs (mail user agents). Examples of MUAs include pine, mutt and Netscape Communicator.

Exim was developed by Philip Hazel at the University of Cambridge for use on Unix systems connected to the Internet.

Exim is an incredibly powerful piece of software, and is able to handle complex mail routing requirements. It can be used as a drop-in replacement for the venerable Sendmail application, which is the standard Unix MTA.

This course will bring you up to speed with the basics of running an Exim based mail system, but you should consult Exim's excellent documentation for further information and help. http://www.exim.org/

You can also find for information in the exim(8) man page, as well as in the "/usr/share/doc/exim/" directory on your local Debian system. This documentation is installed as part of the Exim Debian package.

Exim configuration:

You may remember that during your Debian installation process, you were prompted to answer a few questions to configure your e-mail system. These questions and your answers were processed by the Exim Debian package's installation script, which is configured to run when the Exim package has been installed.

You can choose to re-run this script at any time, by simply running "**`/usr/sbin/eximconfig`**" as root. You can do this now:

```
debian:~# eximconfig
You already have an exim configuration. \
                Continuing with eximconfig
will overwrite it. It will not keep any local \
                 modifications you have made.
If that is not your intention, you should break \
```

```
                out now. If you do continue,
then your existing file will be renamed with \
                .O on the end.
[---Press return---]
```

The configuration files which the script mentions are found in the "/etc/exim" directory. The main configuration file is "/etc/exim/exim.conf".

The script will present you with a set of choices:

```
<!--  =================== -->
You must choose one of the options below:

 (1) Internet site; mail is sent and received directly \
                using SMTP. If your
     needs don't fit neatly into any category, you \
                probably want to start
     with this one and then edit the config file by hand.

 (2) Internet site using smarthost: You receive Internet \
                mail on this
     machine, either directly by SMTP or by running \
                a utility such as
     fetchmail. Outgoing mail is sent using a smarthost. \
                Optionally with
     addresses rewritten. This is probably what you want \
                for a dialup
     system.

 (3) Satellite system: All mail is sent to another machine, \
                called a "smart
     host" for delivery. root and postmaster mail is \
                delivered according
     to /etc/aliases. No mail is received locally.

 (4) Local delivery only: You are not on a network. \
                Mail for local users is delivered.

 (5) No configuration: No configuration will be done now; \
                your mail system
     will be broken and should not be used. You must \
                then do the
     configuration yourself later or run this script, \
                /usr/sbin/eximconfig,
     as root. Look in /usr/share/doc/exim/example.conf.gz

Select a number from 1 to 5, from the list above.
Enter value (default='1', 'x' to restart):
```

If you are setting up the machine to act as a server, but not specifically for e-mail, and you already have a dedicated e-mail server on your network, then you should

select the second option.

If you are setting up the machine to act as a desktop, you should select the third option.

If you are setting up the machine to act as a mail server, then you should select the first option:

```
Select a number from 1 to 5, from the list above.
Enter value (default='1', 'x' to restart): 1
<!-- ====================== -->
What is the 'visible' mail name of your system? \
                This will appear on
From: lines of outgoing messages.
Enter value (default='debian', 'x' to restart):
```

Here you are prompted for the mail domain of your system. If your server was called "server.example.com", then you would probably want to configure your mail name as "example.com"; ie, e-mail addresses will have the form "user@example.com".

```
Enter value (default='debian', 'x' to restart): example.com
<!-- ========================== -->
Does this system have any other names which may \
                appear on incoming
mail messages, apart from the visible name above \
                (example.com) and
localhost?

By default all domains will be treated the same; \
                if you want different
domain names to be treated differently, you will \
                need to edit the config
file afterwards: see the documentation for the \
                "domains" director option.

If there are any more, enter them here, separated \
                with spaces or commas.
If there are none, say 'none'.
Enter value (default='none', 'x' to restart):
```

This question relates to "virtual domains" which you might host on your system. As stated above, the simple configuration system can only handle treating all virtual domains in the same manner, and this is fine for most simple setups.

For example, we may wish to receive e-mail destined for "user@example.org" (_.org_) as well as that destined for "user@example.com":

```
Enter value (default='none', 'x' to restart): example.org
<!-- ============================== -->
```

Again, as stated above, e-mail destined for either domain will be treated equally.

```
All mail from here or specified other local machines \
                to anywhere on
the internet will be accepted, as will mail from \
                anywhere on the
internet to here.

Are there any domains you want to relay mail for\
                ---that is, you are
prepared to accept mail for them from anywhere \
                on the internet, but
they are not local domains.

If there are any, enter them here, separated with \
                spaces or commas. You
can use wildcards. If there are none, say 'none'. \
                If you want to relay
mail for all domains that specify you as an MX, \
                then say 'mx'
Enter value (default='none', 'x' to restart):
```

This question relates to any domains for which we are the backup mailserver. Although you can tell Exim to use MX records to determine this, rather than using a hardcoded list, this is not recommended.

```
Enter value (default='none', 'x' to restart):
<!--
==================== -->
Obviously, any machines that use us as a smarthost \
                have to be excluded
from the relaying controls, as using us to relay mail \
                for them is the
whole point.

Are there any networks of local machines you want \
                to relay mail for?

If there are any, enter them here, separated with \
                spaces or commas.  You
should use the standard address/length format \
                (e.g. 194.222.242.0/24)
If there are none, say 'none'.

You need to double the colons in IPv6 addreses \
```

```
                (e.g.  5f03::1200::836f::::/48)
Enter value (default='none', 'x' to restart):
```

Here you must specify a list of machines and/or networks for which we will relay mail for. Be very careful with your answer here, otherwise you could be allowing untrusted people to use your mail server!

You should normally only specify your own network block:

```
Enter value (default='none', 'x' to restart): 192.168.1.0/24
```

This means that any machines on the 192.168.1.0/24 network will be able to use your machine as an SMTP relay.

```
Enter value (default='none', 'x' to restart): 192.168.1.0/24
Names are localhost:example.com:example.com!
<!--======================== -->
Mail for the 'postmaster' and 'root' accounts is \
                usually redirected
to one or more user accounts, of the actual \
                system administrators.
By default, I'll set things up so that mail for \
                'postmaster' and for
various system accounts is redirected to \
                'root', and mail for 'root'
is redirected to a real user.  This can be \
                changed by editing /etc/aliases.

Note that postmaster-mail should usually be \
                read on the system it is
directed to, rather than being forwarded elsewhere,\
                so (at least one of)
the users you choose should not redirect their \
                mail off this machine.

Which user account(s) should system \
                administrator mail go to ?
Enter one or more usernames separated \
                by spaces or commas .  Enter
'none' if you want to leave this mail in \
                'root's mailbox - NB this
is strongly discouraged.  Also, note that \
                usernames should be lowercase!
```

As directed above, you should direct e-mail for the 'postmaster' and 'root' accounts to a normal user account, preferably your own.

```
Enter value ('x' to restart): student
```

These particular settings can be modified later in the **`/etc/aliases`** file.

Once that's done, you'll be presented with a configuration summary:

```
The following configuration has been entered:
<!--
======================== -->
Mail generated on this system will have 'example.com' used
as the domain part (after the @) in the From: field \
                and similar places.

The following domain(s) will be recognised as \
                referring to this system:
 localhost, example.com, example.com

Mail for postmaster, root, etc. will be sent to student.

Local mail is delivered.

Outbound remote mail is looked up in the Internet \
                DNS, and delivered
using that data if any is found; otherwise such \
                messages are bounced.


Note that you can set email addresses used for \
                outgoing mail by editing
/etc/email-addresses.

Is this OK ?  Hit Return or type 'y' to confirm it \
                and install,
or 'n' to make changes (in which case we'll go \
                round again, giving you
your previous answers as defaults.     (Y/n)
```

If you're happy with the configuration, you can simply hit **Enter**:

```
your previous answers as defaults.     (Y/n) Y

Keeping previous /etc/exim/exim.conf as /etc/exim/exim.conf.O

Keeping previous /etc/aliases as /etc/aliases.O

Keeping previous /etc/mailname as /etc/mailname.O

Configuration installed.
```

```
debian:~#
```

Once you've reached this point, you should inspect the /etc/exim/exim.conf file to familiarise yourself with its contents.

Some of the more important options are:

```
qualify_domain = example.com
```

This specifies the domain which will be appended to any addresses which do not already have one.

```
local_domains = localhost:example.com:example.org
```

This list of domains specifies which @domains are considered to be local. Exim will then attempt to deliver any e-mail to these addresses in the relevant file in /var/spool/mail/${user}, where user is taken from the address "user@domain".

```
relay_domains =
```

This is a list of domains for which we accept e-mail, although we don't consider them to be local domains. These are domains for which we are a backup or secondary mail server.

```
host_accept_relay = 127.0.0.1 : ::::1 : 192.168.1.0/24
```

This is the list of networks for which we will act as an SMTP server, and thus relay mail for.

The **`/etc/aliases`** file contains a simple list in the following format:

```
postmaster: root
root: student
```

In the above example, Exim will redirect e-mail destined for "postmaster" to "root",

and will in turn redirect e-mail for "root" to "student".

You can also specify multiple target addresses, separated with a comma:

```
root: student1, student2
```

Working with Exim:

You can use the "mailq" command to view the current mailq on your Exim system. This command will display all the current e-mail that is in the queue and has yet to be delivered, whether locally or remotely. It will also specify how long the e-mail has been waiting, as well as its size.

You can use the "exiwhat" command to view the current list of Exim processes, as well as what each process is currently doing.

Switching to Sendmail:

Although Debian comes with Exim by default, the standard Unix MTA is Sendmail, and this is the standard for most other Linux distributions. If you are more comfortable with Sendmail, or have some other reason for using it instead of Exim, Debian will allow you to do this.

You can use the Debian package system to remove Exim, and install Sendmail in its place:

```
??????????????????????
```

As you can see, the Exim package has been removed, and the Sendmail one installed in its place. You will now be prompted to fill in the information necessary in order to configure the mail system, similar to what you've been required to do with Exim previously:

```
adduser: Warning: The home dir you specified already exists.
Saving current /etc/mail/sendmail.mc,cf to /var/backups

You are doing a new install, or have erased \
                /etc/mail/sendmail.mc.
If you've accidentaly erased /etc/mail/sendmail.mc, check
/var/backups.
```

```
Sendmail will not start until it is configured.
Do you wish to configure sendmail now, or wait until later?

Configure now ? (y/N)
```

If you specify that you wish to configure Sendmail later, you'll be presented with the following:

```
To configure sendmail later, type sendmailconfig
After configuring sendmail, you can start it via \
                /etc/init.d/sendmail start
Press [ENTER]

debian:~#
```

Configuring Sendmail

Ok, let's now start up the "sendmailconfig" script, and go through the configuration settings: /etc/init.d/sendmail start sendmailconfig.

```
debian:~# sendmailconfig

Sendmail Configuration
----------------------
By answering the following questions, you can \
                configure sendmail for your
system. Default values are determined either by \
                your existing configuration
or from common usage.

Press [ENTER]

Mail Name
---------
Your 'mail name' is the hostname portion of \
                the address to be shown on
outgoing news and mail messages (following \
                the username and @ sign).  This
name will be used by other programs besides \
                sendmail; it should be the single,
full domain name (FQDN) from which mail \
                will appear to originate.

Mail name? [example.com]
```

This is identical to the Exim configuration section in this that we covered previously.

```
Null Client
-----------
A special configuration known as the "null client" \
                can be created for this
host if all mail should be forwarded to a \
                central hub via a local SMTP-based
network. This may be a suitable configuration \
                if you want to forward all of
your mail to your local Internet service \
                provider (ISP) for delivery.

To enable this option, give the name of the \
                host to which all mail should be
forwarded. Otherwise leave the option \
                empty to disable it.
To remove a prior name, use 'NONE'.

Null client forward host? []
```

A "null client" configuration is suitable for a workstation or similar system, which simply needs to redirect all its e-mail to the central mail server on your network. we'll leave this blank for our current configuration.

```
Smart Host
----------
A "Smart Host" is one that can deliver mail to \
                external machines.  By using
a "Smart Host", we don't need DNS, or good \
                connectivity ourselves.  This is
most likely what you want if you have a \
                dialup link, or sit behind a firewall.

To enable this option, give the name of the \
                host to which all non-local mail
should be forwarded.  Otherwise leave \
                the option empty.
To remove a prior name, use 'NONE'.

Smart Host:? []
```

A "smart host" configuration is similar to a "null client", except it also does not require permanent connectivity to the Internet. This is a good choice if you connect to the Internet using a dial-up connection. we'll leave this option blank for our current configuration.

```
Address Canonification
----------------------
Usually sendmail will canonify all addresses by \
```

```
                consulting a name server and
resolving hosts to their fully qualified domain \
                name (FQDN). Under special
circumstances you may want to disable this \
                feature, for example if this
machine acts only as a mail gateway.

Disable address canonification? [N]
```

We will leave address canonification enabled. This is analogous to Exim's "qualify_domain" option.

```
SMTP Mailer
-----------
If you plan to exchange mail with other computers, \
                you should enable the
SMTP mailer. Even if you don't plan to exchange \
                mail with other computers,
it is still a good idea to enable this so local \
                programs can use it.

Enable the SMTP mailer? [Y]
```

As recommended, we will enable the SMTP mailer.

```
Masquerade Envelope
-------------------
If you want mail envelopes (as well as mail \
                headers) to appear to come from
'example.com', you can enable this option.

Masquerade envelopes? [Y]
```

This is useful if your mail server is called "mail.example.com", but you wish for your outgoing e-mail to appear as if it is coming from "example.com". You will normally want to enable this.

```
All Masquerade
--------------
If enabled, this feature will cause recipient \
                addresses to also appear to come
from 'example.com'. Normally they get the \
                local hostname.
Although this may be right for ordinary users, \
                it can break local aliases. For
example, if you send to "localalias", the \
```

```
                originating sendmail will find that
alias and send to all members, but send \
                the message with
"To: localalias@example.com". Since that \
                alias likely does
not exist, replies will fail. Use this feature \
                ONLY if you can guarantee that
the ENTIRE namespace of 'example.com' \
                supersets all the
local entries. If in doubt, it is safe to leave \
                this option disabled.

All masquerade? [N]
```

As recommended, we won't enable "all masquerade".

```

Don't masquerade mail to local users
------------------------------------
Send mail to local recipients without masquerading.

Daunt masquerade local? [N]
```

we'll also leave local masquerading disabled.

```

Always Add Domain
-----------------
If enabled, the local host domain is included \
                even on locally delivered mail.
Normally it is not added unless it is \
                already present.

Always add domain? [N]
```

This means that local only e-mail will not have the machine name or domain name appended to it; this is the default behavior, and should be left as is.

```

Mail Acceptance
---------------
Sendmail is usually configured to accept mail \
                for your mail name
(example.com). However, under special \
                circumstances you
may not wish sendmail to do this, particularly \
                if (and disabling this option
generally requires that) mail for \
                'example.com' is MXed
to another host. If in doubt, it is safe to \
```

```
                leave this option enabled.

Accept mail for 'example.com'? [Y]
```

As directed, it is safe to simply leave this option enabled.

```
Alternate Names
---------------
In addition to the canonical mail name \
                'example.com', you can
add any number of additional alternate \
                names to recognize for receiving mail.
If other hosts are MXed to you for local mail, \
                this is where you should list
them. This list is saved into the file \
                /etc/mail/local-host-names
so it can be changed later as needed.

To answer this question, separate each \
                alternate name with a space, or answer
'NONE' to eliminate all alternate names.

Alternate names? []
```

This option is similar to the "local_domains" option in Exim; it specifies a list of domain names which we consider to be "local" to this system; ie, we will accept and attempt to deliver e-mail destined for user@domain. Sendmail keeps a list of these domains in the "/etc/mail/local-host-names" files.

```
Trusted Users
-------------
Sendmail allows a special group of users to \
                set their envelope "From" address
using the -f option without generating a \
                warning message. If you have
software such as Majordomo installed, you \
                will want to include the usernames
from such software here. Note that "root", \
                "daemon", and "uucp" are included
automatically and do not need to be specified. \
                This list is saved into the
file /etc/mail/trusted-users so it can be \
                changed later as needed.

To answer this question, separate each \
                username with a space, or answer
'NONE' to eliminate all usernames.

Trusted users? []
```

Leave this as the default, unless you have a specific reason to add a trusted user here. You will normally not need to do this, unless you are running mailing list software such as Majordomo.

```
Redirect Feature
----------------
If enabled, this feature will allow you to \
                alias old names to
<new-address>.REDIRECT, causing \
                sendmail to return mail to the sender with
an error but indicating the recipient's new address.

Enable redirect option? [N]
```

This is a nice option to enable if you have a large userbase with a high rate of turnover. we'll leave this option disabled for now though.

```
UUCP Addresses
--------------
Sendmail can be configured to be smart \
                about UUCP addresses, or it can do
nothing special with UUCP addresses at all. \
                If you care about UUCP, you will
need to do some additional configuration, \
                perhaps outside of this script.

*** NOTE *** If you use a smart host or do \
                any kind of forwarding (ie
LUSER_RELAY and LOCAL_RELAY), it is \
                important that you say "Yes"
here to prevent a multi-level relay hole - \
                unless you know for *SURE* that
your smart-host does not deal with UUCP addresses.

(Be safe and just say Y)

Enable UUCP addressing? [Y]
```

UUCP (Unix to Unix Copy Protocol) was the method used for transfering e-mail between Unix systems before the advent of the Internet. It is still very useful for handling e-mail for systems which do not have a permanent Internet connection. It's recommended that you leave this setting on.

```
Sticky Host
-----------
If enabled, mail sent to 'user@example.com' is \
                marked as
"sticky" -- that is, the local addresses aren't \
```

```
                matched against UDB and don't
go through ruleset 5. This is used if you want \
                a setup where 'user' is not
necessarily the same as 'user@example.com', \
                e.g., to make
a distinct domain-wide namespace. \
                If in doubt, it is safe to leave this
option disabled.

Enable sticky host option? [N]
```

As recommended, you can leave this option disabled.

```
DNS
---
If you are directly connected to the Internet and \
                have access to a domain
name server, you should enable this option.

Enable DNS? [Y]
```

If you are configuring a dial-up system, you can disable this option; otherwise, you should always have it enabled.

```
Best MX is Local
----------------
If enabled, this option will cause sendmail to accept \
                mail as though locally
addressed for any host that lists this machine as the \
                best possible MX record.
This generates additional DNS traffic, but should be \
                OK for low-to-medium
traffic hosts. N.B.: This feature is fundamentally \
                incompatible with wildcard
MX records. If you have a wildcard MX record that \
                matches your domain, you
cannot use this feature.

Assume best MX is local? [N]
```

We will leave this disabled for now.

```
Mailertable
-----------
If enabled, this option causes sendmail to read \
                mail routing rules from
the text file /etc/mail/mailertable.  This is needed \
```

```
                for unusual mailers like
ifmail and fax programs.
More information is in \
                /usr/share/doc/sendmail-doc/op/op.txt.gz.

Enable the mailertable feature? [N]
```

You should peruse the documentation found in "/usr/share/doc/sendmail-doc/op/op.txt.gz" to get an idea of what you can do here; but we can leave this disabled for now.

```
Sendmail Restricted Shell
-------------------------
If enabled, this option causes sendmail to use the \
                sendmail restricted shell
program (smrsh) instead of /bin/sh for mailing to \
                programs. This improves your
ability to control what gets run via email; only \
                those programs which appear
in a special directory can be run. If you enable \
                this option, please carefully
read the smrsh(8) man page for further information.

Use the Sendmail Restricted Shell (smrsh)? [Y]
```

This is a desired security option for Sendmail, and should be enabled unless you have a very specific reason not to do so.

```
Mailer Name
-----------
You can change the name used for internally \
                generated outgoing messages.
Usually this is 'MAILER-DAEMON' but it would \
                not be unreasonable to change
it to something such as 'postmaster'.

Mailer name? [MAILER-DAEMON]
```

Leave this as "MAILER-DAEMON".

```
Me Too
------
Sendmail normally excludes the sender address \
                from group expansion.  Enabling
this option will cause the sender to be included.
```

```
Enable me too option? [N]
```

This option is self-explanatory; you can simply leave it at the default.

```
Message Timeouts
----------------
Sendmail will issue a warning message to the \
                sender if it can't deliver a
message within a reasonable amount of time. \
                It will also send a failure
notification and give up trying to deliver the \
                message if it can't deliver it
after an unreasonable amount of time.

You can configure the message timeouts after \
                which warning and failure
notifications are sent. Sendmail's defaults are 4 \
                hours and 5 days (4h/5d),
respectively, but many people feel warnings after \
                only 4 hours are premature.

Message timeouts? [4h/5d]
```

You can leave the values at the default, unless you are wanting to tweak your mail system.

```
Configuration Complete
----------------------
Advanced configuration, such as alternate mailers, \
                the use of mailertables,
Bitnet domains, and UUCP domains can be \
                accomplished by manually editing the
/etc/mail/sendmail.mc configuration file and rerunning
'/usr/sbin/sendmailconfig' to generate the \
                appropriate /etc/mail/sendmail.cf
file. (Local changes made at the end of /etc/mail/sendmail.mc
will be preserved by '/usr/sbin/sendmailconfig'.)
```

Working with Sendmail

Sendmail also uses the **`/etc/aliases`** file, and the format is the same as that for Exim. However, Sendmail keeps a hashed table file of the contents of **`/etc/aliases`**, and you will need to run the "newaliases" command in order for this separate file to be updated after you have edited the original **`/etc/aliases`** file. This tends to be the #1 thing that e-mail administrators forget to do on a

Sendmail system!

As with Exim, you can use the "mailq" command to view the current mailq, and see what mail is still to be delivered.

Index

Symbols

A

B

C

D

E

F

S

T

U

V

W

Z