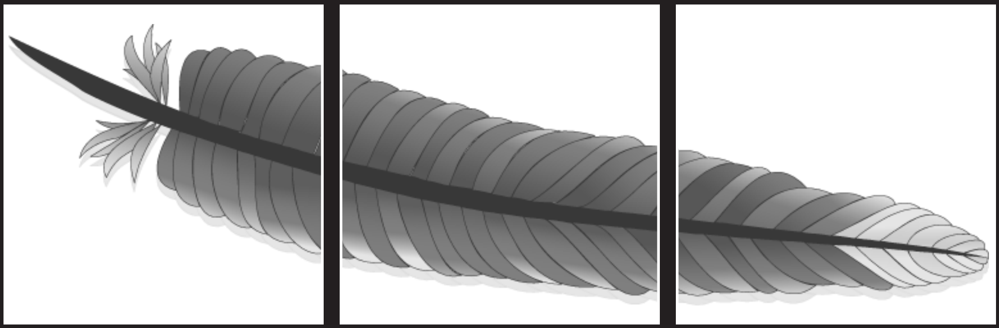
9 SAMPLE APACHE CONFIGURATIONS



This chapter sets up several scenarios and shows you how to best configure Apache to handle each one. For all of the following scenarios, I'm going to create some configuration template files, which will be included in the httpd.conf configuration file. These template files are as follows:

- basic.conf, which will contain the basic configuration options
- modules.conf, which will contain the base set of modules
- logfiles.conf, which will contain the standard log file configuration

You don't have to make these files yourself—I've done this here in order to avoid repeating the same directives over and over again. If you're going to use the following samples, make sure you create these files and place them in your Apache configuration directory. I've included the contents here. Windows users: don't forget to replace the UNIX directory references with the path to your Apache program directory. In the following samples, these files are included using the Include directive.

Several common scenarios follow; since I've already covered the directives used in these sample configurations, I won't go into too much detail here. I'll let you know what chapters to go back and review in each sample. In each sample configuration I've bolded the sections that are unique to the scenario.

The basic.conf file will contain all the global directives, such as ServerRoot, DocumentRoot, and Port. These directives won't change between the different sample configurations.

Windows users will have to replace the paths specified for ServerRoot, LockFile, and PidFile directives with paths to the files within your Apache program folder. The following are examples of how the LockFile, PidFile, and ServerRoot directives might be configured:

LockFile logs\apache.lock

PidFile logs\apache.pid

ServerRoot C:\htdocs\

The ScoreBoardFile directive isn't used under Windows.

```
ServerType standalone
ServerRoot /etc/apache
LockFile /var/lock/apache.lock
PidFile /var/run/apache.pid
ScoreBoardFile /var/run/apache.scoreboard
Timeout 300
Keepalive on
MaxKeepAliveRequests 100
```

```
KeepAliveTimeout 15
MinSpareServers 5
MaxSpareServers 10
StartServers 5
Max Clients 150
MaxRequestsPerChild 100
ExtendedStatus on
Port 80
User www-data
Group www-data
ServerSignature on
AccessFileName .htaccess
DefaultType text/plain
HostNameLookups off
# prevent direct access of .htaccess and .htpasswd files.
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
UseCanonicalName On
TypesConfig /etc/mime.types
<IfModule mod_dir.c>
    DirectoryIndex index.html index.php index.htm index.shtml index.cgi
</IfModule>
```

The modules.conf configuration file contains all modules you're likely to need in the following configurations. Modules not included in this file will be included in these samples.

Many of the modules in this list are left commented—it's a good idea to keep them in, just in case you have to load them in the future.

```
# LoadModule vhost_alias_module /usr/lib/apache/1.3/mod_vhost_alias.so
# LoadModule env_module /usr/lib/apache/1.3/mod_env.so
LoadModule config_log_module /usr/lib/apache/1.3/mod_log_config.so
LoadModule mime_magic_module /usr/lib/apache/1.3/mod_mime_magic.so
LoadModule mime_module /usr/lib/apache/1.3/mod_mime.so
LoadModule negotiation_module /usr/lib/apache/1.3/mod_negotiation.so
LoadModule status_module /usr/lib/apache/1.3/mod_status.so
# LoadModule info_module /usr/lib/apache/1.3/mod_info.so
# LoadModule includes_module /usr/lib/apache/1.3/mod_include.so
LoadModule autoindex_module /usr/lib/apache/1.3/mod_autoindex.so
LoadModule dir_module /usr/lib/apache/1.3/mod_dir.so
LoadModule cgi_module /usr/lib/apache/1.3/mod_cgi.so
# LoadModule asis_module /usr/lib/apache/1.3/mod_asis.so
# LoadModule imap_module /usr/lib/apache/1.3/mod_imap.so
# LoadModule action_module /usr/lib/apache/1.3/mod_actions.so
# LoadModule speling_module /usr/lib/apache/1.3/mod_speling.so
# LoadModule userdir_module /usr/lib/apache/1.3/mod_userdir.so
LoadModule alias_module /usr/lib/apache/1.3/mod_alias.so
LoadModule rewrite_module /usr/lib/apache/1.3/mod_rewrite.so
LoadModule access_module /usr/lib/apache/1.3/mod_access.so
LoadModule auth_module /usr/lib/apache/1.3/mod_auth.so
# LoadModule anon_auth_module /usr/lib/apache/1.3/mod_auth_anon.so
# LoadModule dbm_auth_module /usr/lib/apache/1.3/mod_auth_dbm.so
# LoadModule db_auth_module /usr/lib/apache/1.3/mod_auth_db.so
# LoadModule proxy_module /usr/lib/apache/1.3/libproxy.so
 LoadModule digest_module /usr/lib/apache/1.3/mod_digest.so
# LoadModule cern_meta_module /usr/lib/apache/1.3/mod_cern_meta.so
LoadModule expires_module /usr/lib/apache/1.3/mod_expires.so
# LoadModule headers_module /usr/lib/apache/1.3/mod_headers.so
# LoadModule usertrack_module /usr/lib/apache/1.3/mod_usertrack.so
LoadModule unique_id_module /usr/lib/apache/1.3/mod_unique_id.so
LoadModule setenvif_module /usr/lib/apache/1.3/mod_setenvif.so
# LoadModule sys_auth_module /usr/lib/apache/1.3/mod_auth_sys.so
# LoadModule put_module /usr/lib/apache/1.3/mod_put.so
# LoadModule throttle_module /usr/lib/apache/1.3/mod_throttle.so
# LoadModule allowdev_module /usr/lib/apache/1.3/mod_allowdev.so
# LoadModule eaccess_module /usr/lib/apache/1.3/mod_eaccess.so
# LoadModule php4_module /usr/lib/apache/1.3/libphp4.so
# LoadModule roaming_module /usr/lib/apache/1.3/mod_roaming.so
```

Finally, the logfiles.conf configuration file contains the configuration for the Apache server's log files, as follows:

```
LogLevel warn
LogFormat "%h %l %u %t \"%r\"%c %>s %b \"%{Referer}i\" \"%
➥{User-Agent}i\" %T %v" full
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%
➥{User-Agent}i\" %P %T" debug
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%
➥{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

Basic Apache configuration (with PHP)

It's Tuesday morning. Your boss Jane rushes into your office and tells you that she needs a new web server set up today for a new client.

The client's domain is `www.widgetware.com`, a PHP-based groupware application. There's nothing special about this domain setup, aside from the fact that the client needs PHP installed and running alongside it.

You already have a Linux box available for the server, so you install Apache 1.3 and PHP4.

Sample httpd.conf configuration file

The following is a sample httpd.conf configuration file. I've bolded the sections you should pay attention to.

```
# Load basic Apache configuration
Include basic.conf
# Load Modules
Include modules.conf
# Load the PHP module
LoadModule php4_module /usr/lib/apache/1.3/libphp4.so
AddModule mod_php4.c
AddType application/x-httpd-php .php
# Load Logs
Include logfiles.conf
ServerAdmin webmaster@widgetware.com
ServerName www.widgetware.com
DocumentRoot /var/www/
ErrorLog /var/log/apache/error.log
CustomLog /var/log/apache/ access.log combined
<Directory />
Options Indexes SymLinksIfOwnerMatch MultiViews
    AllowOverride None
</Directory>
<Directory /var/www/>
        DirectoryIndex index.php
    Options Indexes Includes FollowSymLinks MultiViews ExecCGI
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
```

The first highlighted lines in this code show the PHP module being loaded and added to Apache through the LoadModule and AddModule directives. Following this, you configure Apache to recognize the .php file extension as a PHP file.

The next configuration changes come in the <Directory> section. First, you change the DirectoryIndex directive to point to index.php, instead of index.html. Next, you add the ExecCGI option in order to ensure that the PHP scripts will be executed when run.

Slashdotted![1]

You work for Scoot Industries, LLC, which develops ecofriendly transportation. Your company has just released an affordable consumer car powered by vegetable oil, which gets 300 miles to the gallon and doesn't pollute. It has the recommendation of the EPA, Greenpeace, the U.S. Department of Transportation, *and* Michael Moore. It truly is the greatest thing since sliced bread.

And because it's *so* fantastic, it has attracted the attention of news sites everywhere—which of course makes for a lot of traffic coming to your website. As a result, several news organizations have linked to your website, and your poor web server, running under Windows 2000, is working its little heart out.

The problem is that your marketing department has placed several large movie files on the site, in Apple QuickTime format, and they're being requested by just about everybody that's coming to the site. Your dedicated T1 is hitting capacity and connections are being refused when people try to hit the site.

Obviously, you don't want to lose potential customers with a website that's down, but you don't want to be redesigning the site to remove access to the movies either. The secret is to use Apache's mod_rewrite module, to redirect users who are requesting the videos to a page that explains that your website is currently experiencing a heavy amount of traffic, and that the videos aren't available.

Sample httpd.conf configuration file

In the following configuration file, I've bolded the sections that are different from the standard httpd.conf configuration file.

```
# Load basic Apache configuration
Include basic.conf
# Load Modules
Include modules.conf
LoadModule throttle_module /usr/lib/apache/mod_throttle.so
# Load Logs
Include logfiles.conf
ServerAdmin webmaster@scootindustries.com
ServerName www.scootindustries.com
DocumentRoot c:/htdocs
ErrorLog logs/error.log
CustomLog logs/access.log combined
<Directory />
```

[1] Slashdot effect: n. 1. Also known as the "/. effect"; what is said to have happened when a website becomes virtually unreachable because too many people are hitting it after the site was mentioned in an interesting article on the popular Slashdot news service. The term is quite widely used by /. readers, including variants like "That site has been slashdotted again!" —Eric S. Raymond, "The Jargon File, version 4.4.7," home page at `www.jargon.org`.

```
Options Indexes SymLinksIfOwnerMatch MultiViews
    AllowOverride None
</Directory>
<Directory /htdocs/
    Options Indexes Includes FollowSymLinks MultiViews
    AllowOverride All
    Order allow,deny
    Allow from all
    <IfModule throttle_module>
        ThrottlePolicy request 10 1s
        ThrottlePolicy speed 100 1s
    </IfModule>
</Directory>
# Redirect users from the videos to a "Sorry" page
<Directory /htdocs/products/scootbot/videos/>
    Options Indexes Includes FollowSymLinks MultiViews ExecCGI
    AllowOverride All
    RewriteEngine on
    RewriteBase /products/scootbot/
    RewriteRule ^videos/.*
http://www.scootindustries.com/errors/serverbusy.html
</Directory>
```

First, I've enabled bandwidth throttling through the mod_throttle module, which I discussed in Chapter 5. Second, I enabled the module through the LoadModule directive. Third, in the main <Directory> section, I added a throttling policy that limits the server to ten requests and 100 KB per second. This will help to alleviate some of the pressure on the server's resources. I'll likely keep this in place even after the heavy traffic has dissipated, just in case it happens again.

The rewrite rule is here for a temporary measure, however; I want to remove it once the traffic dies down and things get back to normal. As a result, I've placed the rewrite rules in their own <Directory> section, so I can find it easily later.

The bulk of the rewrite configuration comes in the last three lines of the <Directory> statement. First, you need to turn on the rewriting engine through the RewriteEngine directive. Next, you have to tell Apache what root URL the redirected file or directory lives in, using the RewriteBase directive. I covered the modules responsible for redirecting content in Chapter 3.

The last directive is the rewrite rule itself. For the source in this example, I'm using a wild card to specify that all files within the /products/videos directory on the web server will be redirected to the "server busy" page.

Protected intranet directory

Shortly before 3 p.m., Roger saunters over from the human resources office. As he strokes his whiskers, he mentions that they need you to set up an intranet directory on the web server for an employee newsletter. Since the newsletter is going to be talking about successes and failures in the company, it's important that the directory be protected from people outside the network.

OK. Let's assume that addresses in your network follow the 192.168.0.x address format. That is, your IP address on the network might be something like 192.168.0.131, whereas your officemate's IP address might be 192.168.0.132. You'll make use of two Apache features: basic authentication or basic access control.

Since the intranet site is for all employees, the simplest thing to do is use the Order, Allow, and Deny directives to control access to the directory. However, this excludes off-site employees who are checking the site through the Internet. As a result, the best method to use here is a combination of both access control and basic authentication, along with the Satisfy directive.

The Satisfy directive, when set to "any", sets an either/or scenario in Apache; if the user requesting the page is outside the 192.168.0.x IP address range, he'll be presented with a login prompt. Otherwise, the page will be shown to him without having to authenticate on the Apache server.

Sample httpd.conf file

In the following configuration file, I've bolded the sections that are different from the standard httpd.conf configuration file.

```
# Load basic Apache configuration
Include basic.conf
# Load Modules
Include modules.conf
# Load Logs
Include logfiles.conf
ServerAdmin webmaster@staticred.com
ServerName staticred.com
DocumentRoot /var/www/
 ErrorLog /var/log/apache/error.log
CustomLog /var/log/apache/access.log combined
<Directory />
Options Indexes SymLinksIfOwnerMatch MultiViews
    AllowOverride None
</Directory>
<Directory /var/www/>
    Options Indexes Includes FollowSymLinks MultiViews
    AllowOverride All
    Order allow,deny
    Allow from all
```

```
</Directory>
<Directory /var/www/intranet/>
  Options Indexes Includes FollowSymLinks MultiViews ExecCGI
  AllowOverride All
  AuthName "Foo Industries Extranet - Employee Access Only"
  AuthType Basic
  AuthUserFile /var/www/intranet/.htpasswd
  Require valid-user
  Order Allow, Deny
  Allow from 192.168.0
  Deny from all
  Satisfy any
  ErrorDocument 401 /errors/authorization.html
</Directory>
```

In order to maintain this section easily in the future, I've added a special <Directory> section for the protected directory. This will help you find it quicker in the future, should you need to make configuration changes. Because this system is going to be accessed by people outside the company network (hence the name extranet), you won't always know what IP address they're coming from. As a result, you'll want to ask for a username and a password before allowing them access to the site.

The first directive you want to use for configuring authentication is AuthName. This directive will display text in the user's alert box, describing why Apache is requesting that they authenticate before seeing the page. Next, you need to tell Apache what type of authentication to use. As I discussed in Chapter 3, not all browsers support all types of authentication; as a result, you'll want to stick with basic authentication.

After you've told Apache what type of authentication to use, you need to tell it where to find the users allowed on the system. This is the .htpasswd file, which is created through using the htpasswd utility. To bring the authentication together, you finally need to tell Apache that it needs a valid user in order to authenticate.

Because this system is also going to be used inside the network, you don't want to have to force users to log in every time they want to have access to the intranet. As a result, you need to set up an additional access rule based on the IP address of the person visiting.

This is done through the Allow and Deny directives. First, you need to tell Apache the order in which it should read the access rules. In this case, you want it to read the access rules first, followed by the deny rules. This is done through Order, Allow, and Deny. Next, you want to tell Apache which IP addresses it should allow to access the directory. Since everyone in the internal network has an IP address starting with 192.168.0 (for example, 192.168.1.42), you'll set this up with a wild card: Allow from 192.168.0. Next, you have to tell Apache to deny all other IP addresses through the Deny from all directive.

Finally, you have to tell Apache that if either the IP address matches or the user enters the right username and password, the user is allowed access to the directory. This is done through the Satisfy any directive.

Running virtual hosts

Jane is back again. It turns out that the client is running three domains, two of which share the same content. The first two domains are www.widgetware.com and www.widgetware.net, which is a site for their WidgetWare product. They also run www.widgetmasters.org, a community support site for their product. Their resources are limited, so they want to share a single server among all of the sites.

For this setup, you'll need to create <VirtualHost> sections for each of the domains and make use of the ServerAlias directive.

Sample httpd.conf configuration file

```
# Load basic Apache configuration
Include basic.conf
# Load Modules
Include modules.conf
# Load Logs
Include logfiles.conf
ServerAdmin webmaster@widgetware.com
ServerName widgetware.com
ServerAlias widgetware.net
DocumentRoot /var/www/widgetware/
ErrorLog /var/log/apache/widgetware-error.log
CustomLog /var/log/apache/widgetware-access.log combined
<Directory />
Options Indexes SymLinksIfOwnerMatch MultiViews
    AllowOverride None
</Directory>
<Directory /var/www/>
    Options Indexes Includes FollowSymLinks MultiViews
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
NameVirtualHost 204.174.19.10:80
# widgetmasters.org's configuration
<VirtualHost 204.174.19.10>
  ServerName widgetmasters.org
  ServerAdmin webmaster@widgetmasters.org
  DocumentRoot /var/www/widgetmasters
  ErrorLog /var/log/apache/widgetmasters-error.log
  CustomLog /var/log/apache/widgetmasters-access.log
  <Directory /var/www/widgetmasters>
        Options Indexes Includes FollowSymLinks MultiViews
        AllowOverride All
  </Directory>
  ScriptAlias /cgi-bin/ /var/www/widgetmasters/cgi-bin/
  <Directory /var/www/widgetmasters/cgi-bin/>
    Options Indexes, Includes, FollowSymLinks, Multiviews, ExecCGI
    AllowOverride All
  </Directory>
</VirtualHost>
```

As I discussed in Chapter 4, virtual hosts are configured separately from the main domain by using the <VirtualHost> section. In some cases, however, you may need to preface these sections with the NameVirtualHost directive. Other than the <VirtualHost> section heading, however, configuring a virtual host is identical to configuring the main domain name. You still create <Directory> sections within it to specify the options for the main directory, and you configure log files for the virtual hosts.

For more information about virtual hosts, check out Chapter 4.

Compressed HTTP sessions

One of Apache's great features is HTTP compression. This feature, as you may have guessed, compresses the HTML, graphics, and other web files before sending them over the Internet to the browser, which then decompresses the files before displaying them. Many servers have this enabled to save on bandwidth costs. Although you may only save 2 or 3 KB per transfer, it all adds up rather quickly!

Sample httpd.conf configuration file

```
# Load basic Apache configuration
Include basic.conf
# Load Modules
Include modules.conf
# Load Logs
Include logfiles.conf
ServerAdmin webmaster@widgetware.com
ServerName www.widgetware.com
ServerAlias www.widgetware.net
DocumentRoot /var/www/widgetware/
<Directory />
Options Indexes SymLinksIfOwnerMatch MultiViews
    AllowOverride None
</Directory>
<Directory /var/www/>
    Options Indexes Includes FollowSymLinks MultiViews
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
<IfModule mod_dir.c>
    DirectoryIndex index.html index.php index.htm index.shtml index.cgi
```

```
</IfModule>
<IfModule mod_mime.c>
    AddEncoding x-compress Z
    AddEncoding x-gzip gz tgz
    AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
    AddType application/x-httpd-php .php
    AddType application/x-httpd-php-source .phps
    AddType application/x-tar .tgz
    <IfModule mod_gzip.c>
      mod_gzip_on                    Yes
      mod_gzip_temp_dir              /tmp/
      mod_gzip_on                    Yes
      mod_gzip_dechunk               Yes
      mod_gzip_minimum_file_size     300
      mod_gzip_maximum_file_size     0
      mod_gzip_maximum_inmem_size    100000
      mod_gzip_keep_workfiles        No
      mod_gzip_add_header_count      No
      mod_gzip_item_include   file    \.htm$
      mod_gzip_item_include   file    \.html$
      mod_gzip_item_include   mime    text/.*
      mod_gzip_item_include   file    \.html$
      mod_gzip_item_include   file    \.jsp$
      mod_gzip_item_include   file    \.php$
      mod_gzip_item_include   file    \.pl$
      mod_gzip_item_include   mime    ^text/.*
      mod_gzip_item_include   mime    ^application/x-httpd-php
      mod_gzip_item_include   mime    ^httpd/unix-directory$
      mod_gzip_item_include   handler ^perl-script$
      mod_gzip_item_include   handler ^server-status$
      mod_gzip_item_include   handler ^server-info$
      mod_gzip_item_exclude   file    \.css$
      mod_gzip_item_exclude   file    \.js$
      mod_gzip_item_exclude   mime    ^image/.*
    </IfModule>
    AddType image/bmp .bmp
    AddHandler cgi-script .cgi .sh .pl
AddType text/html .shtml
AddHandler server-parsed .shtml
</IfModule>
```

It looks like there's a lot going on in this sample, but there really isn't. First, you want to make sure that the mod_mime module is enabled; if it isn't, you can't add the compressed file types. Next, you need to set up the items that should and shouldn't be compressed on the server. Items that should be compressed are included through the mod_gzip_item_include directive, while items that shouldn't be compressed are added through the mod_gzip_item_exclude directive. In the previous case, you want all HTML and text-based files to be compressed, and all image, CSS, and JavaScript files to not be compressed.

Summing it up

The previous samples should give you a heck of a good head start toward configuring your Apache server. Feel free to mix and match the previous configurations to suit your own purposes. Experimenting with your Apache configuration is one of the best ways to learn (just make sure you have backups of the httpd.conf configuration file!).