To Configure Apache 2

It is important to read the documentation distributed together with the Apache server. These documents are usually kept in directory "<APACHE\_HOME>\manual" or "<APACHE\_HOME>\htdocs\manual" (where <APACHE\_HOME> denotes your Apache's installed directory). Read the tutorials and How-To's.

To install Apache 2, read "[How to install Apache 2](https://www3.ntu.edu.sg/home/ehchua/programming/howto/Apache_HowToInstall.html)". I shall assume that Apache HTTP server is installed in d:\myProject\apache2, running in port 8000. The document root directory is "<APACHE\_HOME>\htdocs".

**1.  Basic Configuration**

Apache is configured by placing configuration directives, such as Listen and ServerName, into a configuration file, which will be read by the Apache executable during the startup.

The default configuration file is called "httpd.conf" (or "apache2.conf") in the directory "<APACHE\_HOME>\conf". Browser through this configuration file.

At a minimum, you need to check the following directives:

* **Listen**: to bind Apache to specific IP addresses and/or ports. HTTP server, by default, runs on port 80 for production. For testing, you could choose a port number between 1024 to 65535, which is not used by an existing application (you can run command "netstat" to check the existing connections). We shall run the Apache at port 8000.
* # Listen: Allows you to bind Apache to specific IP addresses and/or ports.

Listen 8000

* **ServerName**: Set to your DNS hostname, or IP address (to find out your IP address, run command "ipconfig"), or your computer name, or "localhost" (localhost is meant for local loop-back testing only, you can also use the localhost's IP address 127.0.0.1), followed by the port number chosen above.
* # ServerName gives the name and port that the server uses to identify itself.
* # If your host doesn't have a registered DNS name, enter its IP address here.

ServerName *YourHostNameOrIPAddres*:8000

* **ServerRoot**: the Apache installed directory "<APACHE\_HOME>", e.g.,
* # ServerRoot: The top of the directory tree under which the server's
* # configuration, error, and log files are kept.
* # Assume that Apache HTTP server is installed in "d:\myProject\apache2"

ServerRoot "d:/myProject/apache2"

You should use Unix-style forward slash (/) as the *directory separator*, instead of Windows-style backward slash (\) in the configuration file.

* **DocumentRoot**: the document root directory, i.e., home directory of the server. It is set to "<APACHE\_HOME>\htdocs" by default.
* # DocumentRoot: The directory out of which you will serve your documents.
* DocumentRoot "**d:/myPorject/apache2/htdocs**"
* # Access Control for the document base directory
* <Directory "**d:/myPorject/apache2/htdocs**">
* # Show directory listing, and allow symbolic links
* Options Indexes FollowSymLinks
* # Cannot override with .htaccess files.
* AllowOverride None
* # Controls who can get stuff from this server.
* Order allow,deny
* Allow from all

</Directory>

**Caution**: You MUST do a global search on "htdocs", before modifying the document root directory.

**2.  Access Control in Apache HTTP Server**

*Access control* deals with controlling access to a resource, which could a set of directories, files or locations. Access control can be based on the client's identity, which is called *authentication* (discussed in "[HTTP Authentication](https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Authentication.html)"). Access control could also be based on other criteria, such as the network address, the time of day, the browser which the client is using, the types of request methods, and etc.

**2.1  Directory Access Control**

This section deals with access control to directories. The following sections will deal with access control to files and locations.

**<Directory>...</Directory>**: can be used to apply access control to a set of directories. The syntax is:

<Directory *directories*>

# access control directives for the matching directory(ies).

......

</Directory>

The <directory> block directive encloses a set of access-control directives, which will be applied to the matched directory(ies) and its sub-directories. The *directories* specifies the directories applicable to this block. Wildcard can be used in matching: "?" matches exactly one character; "\*" matches zero or more characters; [...] can be used to specify a range of characters, e.g. [c-f]. Extended regular expression (regexe) can be used, which begins with a "~".

**Options**: controls what kinds of actions are permitted for the set of resources under control.

Option [+|-]*option-1* [+|-]*option-2* ...

The available options are:

* Indexes: If the client requests for a directory and there is no indexing file (e.g., "Index.html") in the directory, then the server will return a listing of the directory. If "Indexes" option is disabled, the server returns error "403 Forbidden".
* ExecCGI: Allow execution of CGI script.
* Includes: Allow Server-Side Include (SSI).
* IncludesNOEXEC: Allow SSI, but disable #exec command and #exec CGI.
* FollowSymLinks: Follow symbolic links.
* SymLinksIfOwnerMatch: Follow symbolic links only if the owner is the same.
* MultiViews: Allow content negotiation, such as language negotiation.
* None: Nothing.
* All: All options except MultiViews. This is the default setting.
* + (or -) adds (or removes) that particular option, relative to the current setting. All the other options remain the same. For example, "Option +Indexes -ExecGGI" directive adds the "Indexes" option and removes the "ExecCGI" option from the current setting. The other options remain unaffected.

If no Options directive is used, the effect is All except MultiViews. However, if an Options directive is used without +/-, e.g., "Options Indexes", only Indexes option is available, and the rest of options are off. If +/- is used, only that particular option is changed, the rest of the options remain the same (inherited from the setting at the higher level).

**Example 1**

<Directory **/www**>

Options Indexes ExecCGI

</Directory>

<Directory **/www/sales**>

Options Indexes

</Directory>

<Directory **/www/support**>

Options -Indexes

</Directory>

Since the <Directory> matching applies to sub-directories, "/www" has options Indexes and ExexCGI, "/www/sales" has option Indexes only (the setting in the parent directory is ignored), and "/www/support" has option ExecCGI (inherited from its parent directory).

**Order**: specifies the order in which Allow and Deny directives are evaluated.

Order Deny,Allow | Allow,Deny

* Deny,Allow: Access is allowed by default, and the Deny directives are evaluated before the Allow directives. Any client which does not match a Deny directive "or" does match an Allow directive will be allowed access to the server. (The client is allowed access if it is in both Deny and Allow list, as Allow is evaluated last.)
* Allow,Deny: Access is denied by default, and the Allow directives are evaluated before the Deny directives. Any client which does not match an Allow directive "or" does match a Deny directive will be denied access to the server. (A client in both Allow and Deny will be denied access, as Deny is evaluated last.)

**Apache 2.4**

Apache 2.4 uses a new module called mod\_authz\_host for access control. Instead of the Order, Allow and Deny directives in Apache 2.2, it uses a new directive Require. For example,

// Apache 2.2

Order deny,allow

Deny from all

// Apache 2.4

Require all denied

// Apache 2.2

Order allow,deny

Allow from all

// Apache 2.4

Require all granted

// Apache 2.2

Order Deny,Allow

Deny from all

Allow from example.com

// Apache 2.4

Require host example.org

[TODO] Tidy up the following examples.

**Example 2**

Order Deny,Allow

Deny from all

Allow from test101.com

1. access is allowed by default;
2. all hosts are denied;
3. those in the domain "\*.test101.com" are allowed.

Consequently, only hosts in "\*.test101.com" are allowed.

**Example 3**

Order Allow,Deny

Allow from test101.com

Deny from sales.test101.com

1. access is denied by default;
2. all hosts in the "\*.test101.com" domain are allowed, and;
3. hosts in the "\*.sales.test101.com" sub-domain are denied.

Consequently, all hosts in the "\*.test101.com" domain except "\*.sales.test101.com" are allowed.

On the other hand, if the Order is changed to Deny,Allow, all hosts will be allowed access (by default). This happens because, regardless of the actual ordering of the directives in the configuration file, the Allow from test101.com will be evaluated last and will override the Deny from sales.test101.com. Any other hosts are allowed access by default.

**Example 4**

<Directory /home>

Order Allow,Deny

</Directory>

Access is denied to all hosts to directory "/home", based on the default setting.

**Example 5**

<Directory /home>

Order Deny,Allow

Deny from all

</Directory>

Access is denied to all hosts to directory "/home". Although the access is allowed by default, Deny from all prohibits all hosts.

**Allow**: specifies which hosts can access a set of resources. Access can be controlled by hostname, IP Address, IP Address range, or environment variables.

Allow from all|host|env=*env-variable*

If Allow from all is specified, then all hosts are allowed access, subject to the configuration of the Deny and Order directives. To allow only particular hosts or groups of hosts to access the server, the host can be specified in any of the following formats:

* Domain-name: Hosts whose names match, or end in, this string are allowed access. For example, Allow from test101.com will match sales.test101.com and support.test101.com but it will not match www.test999.com.
* Full/partial IP address: For example, Allow from 10.1 grants access to all IP addresses in the form 10.1.\*.\*.
* A network/netmask pair: For example, Allow from 10.1.0.0/255.255.0.0.

If Allow from env=*env-variable* is specified, then the request is granted if the environment variable *env-variable* exists. This directive can be used to allow access based on such factors as the clients User-Agent (browser type), Referer, request method, or other HTTP request header.

**Example 6**

SetEnvIf User-Agent ^Mozilla/4.0 Mozilla4\_browser

<Directory /docroot>

Order Deny,Allow

Deny from all

Allow from env=Mozilla4\_browser

</Directory>

In this example, browsers with a User-Agent string beginning with Mozilla/4.0 will be allowed access. All other type of browsers will be denied.

**Deny**: restricts access based on hostname, IP address, or environment variables.

Deny from all|host|env=*env-variable*

The arguments for the Deny directive are identical to the arguments for the Allow directive.

**File .htaccess**

In each directory, you can create a file called ".htacces" to control the access into that particular directory, if AllowOverride is turned on. The directives inside the .htaccess override the <directory> directive. The relevant directives to enable .htaccess in "httpd.conf" are:

# AccessFileName specifies the name of the file to look for

# in each directory for access control information.

AccessFileName .htaccess

# AllowOverride controls which options the .htaccess files in

# directories can override. Can be "None", "All", or any

# combination of "Options", "FileInfo", "AuthConfig", and "Limit".

AllowOverride None|All|Options|FileInfo|AuthConfig|Limit

# Prevent files beginning with ".ht" (such as .htaccess, .htpasswd

# from being viewed by clients for security reason.

# Since .htaccess files often contain authorization information.

<Files ~ "^\.ht">

Order allow,deny

Deny from all

Satisfy All

</Files>

# To protect only .htaccess

<Files .htaccess>

Order allow,deny

Deny from all

</Files>

Using .htaccess can prevent frequent re-starting of the server. This is because the configuration directives in "httpd.conf" is read at startup. Any change requires a re-start. The .htaccess is check at each access. Change will take effect for the subsequent accesses. The disadvantage is degradation in performance as the .htaccess has to be check for every access into the directory.

**<Limit *methods*> & <LimitExcept *methods*>**:

<Limit *request-method-1* *request-method-2*>

...*directives*...

</Limit>

Access controls are normally effective for all the request methods (such as GET, POST, HEAD, PUT, DELETE). <Limit> and <limitExcept> blocks can be used to restrict access controls based on the HTTP request method used in the incoming request. This is useful if you have implemented PUT request but wish to limit PUT requests but not GET requests; or you might want to allow GET/HEAD but limit PUT/DELETE.

For <limit>, access control is applied to those methods listed; all the other methods are unrestricted, for example,

<Limit POST PUT DELETE>

Order deny,allow

Deny from All

</Limit>

Access control applied to the methods POST, PUT, and DELETE; all other methods are unrestricted.

The method names listed can be one or more of: GET, POST, PUT, DELETE, CONNECT, OPTIONS, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, and UNLOCK. If GET is used it will also restrict HEAD requests. The TRACE method cannot be limited.

<LimitExcept> is used to enclose a group of access control directives which will be applied to any HTTP access method NOT listed; i.e., it is the opposite of a <Limit> block and can be used to control both standard and nonstandard/unrecognized methods. A <LimitExcept> block should be used in preference to a <Limit> block when restricting access, since a <LimitExcept> block provides protection against arbitrary methods. For example,

<LimitExcept GET POST>

Order deny,allow

Deny from all

</LimitExcept>

Request methods other than GET and POST, such as PUT, DELETE will not be permitted.

**Example 7**

<Directory "d:/myPorject/apache2/users">

AllowOverride FileInfo AuthConfig Limit

Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec

<Limit GET POST OPTIONS PROPFIND>

# Default access is deny

Order allow,deny

# But allow access from all

Allow from all

</Limit>

<LimitExcept GET POST OPTIONS PROPFIND>

# Default access is allow

Order deny,allow

# But deny access from all

Deny from all

</LimitExcept>

</Directory>

**2.2  File Access Control**

<Files file-name>

......

</Files>

Unlike <directory>, file-name is relative to the DocumentRoot.

(Under construction)(Give some examples)

**2.3  Location Access Control**

<Location URL>

......

</Location>

Limit the scope of directives defined within the block to those matching URL(s).

(Under construction)(Give some examples)

**3.  Virtual Hosts**

Very often, your web server has to support a few hostnames (e.g., www.test101.com, www.test102.com, and etc.), a few IP addresses (with multiple network cards) or listening to a few ports. It is rather unusual and messy to run one server for each of the hostnames, IP addresses, or ports. It is better to run many "virtual hosts" within a single physical web server.

HTTP/1.1 introduces a new feature called "virtual host", which allows you to running multiple hostnames on the same physical server/machine. HTTP/1.1-compliant server can support many hostnames/IP addresses/Ports within one single server. On the other hand, HTTP/1.0 server supports only one TCP address and one host name. In HTTP/1.1, the "Host" request header is mandatory to select one of the virtual hosts.

Read "Virtual Host - How-to" in "htdocs\manual\programs\vhosts\index.html.html"

Apache support (a) Name-based virtual hosts, (b) IP-based virtual hosts, and (c) Port-based virtual host.

**3.1  Named-based Virtual Hosts**

Name-based virtual hosting is usually simpler, since you only need to configure your DNS server to map each hostname to the same IP address and then configure the Apache HTTP Server to recognize the different hostnames. Name-based virtual hosting also eases the demand for scarce IP addresses. Name-based virtual hosting should be used unless there is a specific reason to choose IP-based virtual hosting.

To use name-based virtual hosting, you must designate the IP address (and possibly port) on the server that will be accepting requests for the hosts. This is configured using the NameVirtualHost directive. In the normal case where any and all IP addresses on the server should be used, you can use \* as the argument to NameVirtualHost.

The next step is to create a <VirtualHost> block for each different host that you would like to serve. The argument to the <VirtualHost> directive should be the same as the argument to the NameVirtualHost directive (i.e., an IP address, or \* for all addresses). Inside each <VirtualHost> block, you will need at minimum a ServerName directive to designate which host is served and a DocumentRoot directive to show where in the file system the content for that host lives.

If you are adding virtual hosts to an existing web server, you must also create a <VirtualHost> block for the existing host. The ServerName and DocumentRoot included in this virtual host should be the same as the global ServerName and DocumentRoot. List this virtual host first in the configuration file so that it will act as the default host.

For example, suppose that you are serving the domain www.test101.com and you wish to add the virtual host www.test102.com, which resolves to the same IP address. Then you simply add the following to "httpd.conf":

# Virtual host for all IP addresses at Port 80

NameVirtualHost \*

# First virtual host shall be the main server, the default host.

<VirtualHost \*>

ServerName www.test101.com

DocumentRoot /www101

</VirtualHost>

<VirtualHost \*>

ServerName www.test102.com

DocumentRoot /www102

</VirtualHost>

You can alternatively specify an explicit IP address in place of the \* in both the NameVirtualHost and <VirtualHost> directives, if your server accepts multiple IP addresses.

Many servers want to be accessible by more than one name. This is possible with the ServerAlias directive, placed inside the <VirtualHost> section. For example if you add this to the first <VirtualHost> block above

ServerAlias www.test101.com \*.test101.com

then requests for all hosts in the test101.com domain will be served by the www.test101.com virtual host. The wildcard characters "\*" and "?" can be used to match names. Of course, you can't just make up names and place them in ServerName or ServerAlias. You must first have your DNS server properly configured to map those names to an IP address associated with your server.

Now when a request arrives, the server will first check if it is using an IP address that matches the NameVirtualHost. If it is, then it will look at each <VirtualHost> section with a matching IP address and try to find one where the ServerName or ServerAlias matches the requested hostname. If it finds one, then it uses the configuration for that server. If no matching virtual host is found, then the first listed virtual host that matches the IP address will be used.

As a consequence, the first listed virtual host is the default virtual host. The DocumentRoot from the main server will never be used when an IP address matches the NameVirtualHost directive. If you would like to have a special configuration for requests that do not match any particular virtual host, simply put that configuration in a <VirtualHost> container and list it first in the configuration file.

IP-based virtual hosts use the IP address of the connection to determine the correct virtual host to serve. Therefore you need to have a separate IP address for each host. With name-based virtual hosting, the server relies on the client to report the hostname as part of the HTTP headers. Using this technique, many different hosts can share the same IP address.

For testing virtual host without access to DNS server: You can create a few hostnames pointing to your own IP address (or localhost) in your local DNS lookup table "hosts". For example:

192.123.123.1 www.yellow.com

192.123.123.1 www.sales.yellow.com

192.123.123.1 www.orange.com

127.0.0.1 localhost

127.0.0.1 apple88

127.0.0.1 orange99

In Windows, the local DNS lookup table is called "%SYSTEM\_ROOT%\system32\drivers\etc\Hosts".

**3.2  IP-based Virtual Hosts**

As the term IP-based indicates, the server must have a different IP address for each IP-based virtual host. This can be achieved by the machine having several physical network connections, or by use of virtual interfaces which are supported by most modern operating systems (see system documentation for details, these are frequently called "ip aliases", and the "ifconfig" command is most commonly used to set them up).

For example:

<VirtualHost 192.123.10.1>

DocumentRoot /www201

ServerName www.test201.com

</VirtualHost>

<VirtualHost 192.123.10.2>

DocumentRoot /www202

ServerName www.test202.com

</VirtualHost>

Host can be \_default\_, in which case it matches anything no <VirtualHost> matches.

**3.3  Port-based Virtual Hosts**

Use different port number for different virtual hosts. The advantage is you do not need many domain names or IP addresses. However, the client may not be familiar with the format of accessing HTTP server with a non-default port number.

An example is as follows:

Listen 80

<VirtualHost \*:80>

ServerName localhost:80

DocumentRoot /var/www1

ErrorLog <APACHE\_HOME>/error.log

CustomLog <APACHE\_HOME>/access.log combined

# Set the permission of the document base directory

<Directory "/var/wwwl">

Options Indexes FollowSymLinks MultiViews

AllowOverride None

# Apache 2.4

Require all granted

# Apache 2.2

# Order allow,deny

# allow from all

</Directory>

</VirtualHost>

Listen 8080

<VirtualHost \*:8080>

ServerName localhost:8080

DocumentRoot /var/www2

ErrorLog <APACHE\_HOME>/error.log

CustomLog <APACHE\_HOME>/access.log combined

# Set the permission of the document base directory

<Directory "/var/www2">

Options Indexes FollowSymLinks MultiViews

AllowOverride None

# Apache 2.4

Require all granted

# Apache 2.2

# Order allow,deny

# allow from all

</Directory>

</VirtualHost>

The Listen directive tells the Apache which port to listen to. Apache can listen to more than one port by using multiple Listen directives.

**4.  Setup HTTPS for Apache Server (Windows)**

For Ubuntu, read "[Setup HTTPS for Apache (Ubuntu)](https://www3.ntu.edu.sg/home/ehchua/programming/howto/Ubuntu_HowTo.html#https)".

**Step 1: Create a Certificate for the Web Server**

The first step to set up SSL support is to create a certificate for your web server. To do so, you need OpenSSL, which is an open-source software available at [http://www.openssl.org](http://www.openssl.org/). Apache's Windows binary package includes OpenSSL in "<APACHE\_HOME>\bin".

Issue the following command to create a *self-signed certificate* for the server. First of all, a public-private key pair is generated. The private key is saved in "MyServer.key" (which shall be kept in a secure location). The public key is saved in a certificate "MyServer.crt" to be transferred to the user.

> **openssl req -x509 -days 36500 -newkey rsa:2048 -nodes -keyout MyServer.key -out MyServer.crt**

**-subj /C=SG/O=MyCompany/CN=localhost**

// If error "Unable to load config info from /usr/local/ssl/openssl.cnf" encountered

> **openssl req -x509 -days 36500 -newkey rsa:2048 -nodes -keyout MyServer.key -out MyServer.crt**

**-subj /C=SG/O=MyCompany/CN=localhost** **-config ../conf/openssl.cnf**

The option are:

* -x509: requests a X.509 certificate to be generated.
* -days 36500: sets the expiration period for the certificate. The default is 30 days. I set to 100 years.
* -newkey rsa:2048: generate a new key-pair, using RSA of bit-length 2048.
* -nodes: no passphrase is to be used for the private key file.
* -keyout and -out: specify the output private key-file and certificate.
* -subj sets the country code (/C), company name (/O), and the common name (/CN). If you leave these out, you'll be prompted for them. The CN (Common Name) must be the same as your ServerName in your Apache configuration, otherwise the certificate won't match and users will receive a warning when connecting.
* -config <openssl.conf>: specify the openssl config file.
* Refer to <http://www.modssl.org/docs/2.2/ssl_reference.html> for more information about OpenSSL command syntax.

To view the content of a certificate (which contains the public key of the server), issue the following openssl command:

> **openssl x509 -in server.crt -noout -text**

**Step 2: Configuring Apache HTTP Server**

First of all, move the private key file (MyServer.key) and certificate (MyServer.crt) to the Apache's configuration directory (<APACHE\_HOME>/conf).

In apache's main configuration "httpd.conf" (under <APACHE\_HOME>/conf), check the following directives:

# Load the SSL module

LoadModule ssl\_module modules/mod\_ssl.so

# Include ssl configuration from an external file

Include conf/extra/httpd-ssl.conf

The LoadModule loads the SSL module and the Include directive includes more configuration options for SSL support in "conf/extra/httpd-ssl.conf", as follows

# Use HTTPS default port number of 443

Listen 443

# Define a Virtual Host for HTTPS under directory "wwwssl"

<VirtualHost \*:443>

DocumentRoot "<APACHE\_HOME>/wwwssl"

ServerName localhost:443

ErrorLog "<APACHE\_HOME>/logs/error.log"

TransferLog "<APACHE\_HOME>/logs/access.log"

SSLEngine on

SSLProtocol all -SSLv2

SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5

SSLCertificateFile "<APACHE\_HOME>/conf/MyServer.crt"

SSLCertificateKeyFile "<APACHE\_HOME>/conf/MyServer.key"

# Set the permission of the document base directory

<Directory "<APACHE\_HOME>/wwwssl">

Options Indexes FollowSymLinks MultiViews

AllowOverride None

Order allow,deny

allow from all

# SSLRequireSSL

</Directory>

</VirtualHost>

**Verifying SSL Installation**

Create the document root directory "wwwssl", and place a welcome page (e.g., index.html).

Start the Apache Server. Start a browser and issue https://localhost.

Because the server certificate is self-signed and not signed by a trusted CA (Certificate Authority), browser issues a warning. Accept the warning and continue...

**What if...**

In case of error in the installation:

* Check the Apache and SSL log.
* Try connecting to the Apache server via OpenSSL as follows:

> **openssl s\_client -connect localhost:443**

If the connection succeeds then an HTTP command such as "`GET /"' to retrieve a web page.

**Password-Protected Private Key File**

You can attached a passphrase (i.e., password) to the private key file. However, to start Apache, you need to either hardcode the passphrase in the apache's configuration file (same security exposure as no passphrase) or provide the passphrase during the start-up dialog (this means that you can't automate the Apache start-up!).

**CA-Signed Certificate**

To generate a certificate for signning by CA:

1. Generate a public-privage key pair and a certificate request:
2. > **openssl req -new -newkey rsa:2048 -nodes**
3. **-keyout MyServer.key -out www.mysite.com.csr**

**-subj /O=MyCompany/OU=MyDepartment/CN=www.mysite.com**

we didn't use the -x509 switch. The command will therefore generate a public-private key pair and certificate request in a .csr file, but not a certificate (.crt file).

1. Send that certificate request file "www.mysite.com.csr" to the CA (with your payment). You may be able to get a free certificate from CAcert.org.
2. Rename the received certificate to MyServer.crt and verify its contents:

> **openssl verify -CAfile /path/to/trusted\_ca.crt -purpose sslserver server.crt**

Check that the certificate corresponds to your private key:

> **openssl x509 -noout -modulus -in server.pem | openssl sha1**

> **openssl rsa -noout -modulus -in server.key | openssl sha1**

1. Install your private key (MyServer.key) and certificate (Myserver.crt) in your apache configuration.

**5.  Miscellaneous Configurations**

**5.1  Log Files**

Apache produces these log files: error log, access log. The default configuration puts the error log in "$APACHE\_home\logs\error.log" and access log in "$APACHE\_home\logs\access.log". Take a quick glance into these log files.

**Error Log**: The configuration directives related to error logging are ErrorLog and LogLevel:

* ErrorLog directive specifies the location of the error log file. For example:
* # ErrorLog: The location of the error log file.
* # If you do not specify an ErrorLog directive within a <VirtualHost>
* # container, error messages relating to that virtual host will be
* # logged here. If you \*do\* define an error logfile for a <VirtualHost>
* # container, that host's errors will be logged there and not here.

ErrorLog logs/error.log

* LogLevel directive controls the types of error messages written to the error log. For example:
* # LogLevel: Control the number of messages logged to the error\_log.
* # Possible values include: debug, info, notice, warn, error, crit,
* # alert, emerg.

LogLevel warn

Sample entries in the error log are as follows:

[Sun Oct 18 16:53:40 xxxx] [error] [client 127.0.0.1] Invalid method in request get /index.html HTTP/1.0  
[Sun Oct 18 18:36:20 xxxx] [error] [client 127.0.0.1] File does not exist: d:/myPorject/apache2/htdocs/t.html

[Sun Oct 18 19:58:41 xxxx] [error] [client 127.0.0.1] client denied by server configuration: d:/myPorject/apache2/htdocs/forbidden/index.html

**Access Log**: The configuration directives related to access logging are CustomLog and LogFormat:

* The CustomLog directive specifies the locations of the access log files. There are 3 types of access logs: common, referrer and agent. Common access log captures client access. Referrer access log captures the "referrer" (as in the Referer request header) of the request. (Referrer can be used for access control or accounting purpose such as e-advertising.) Agent access log captures the types of the browsers used in issuing the request (as in the User-Agent request header). Most installations do not need the referrer and agent access logs. For example:
* # The location and format of the access logfile (Common Logfile Format).
* # If you do not define any access logfiles within a <VirtualHost>
* # container, they will be logged here. Contrariwise, if you \*do\*
* # define per-<VirtualHost> access logfiles, transactions will be
* # logged therein and \*not\* in this file.
* CustomLog "logs/access.log" common
* #CustomLog logs/referer.log referer

#CustomLog logs/agent.log agent

You can combine all the 3 access logs into a single log file, using keyword "combined".

# If you prefer a logfile with access, agent, and referer information

# (Combined Logfile Format) you can use the following directive.

#CustomLog "logs/access.log" combined

* LogFormat directive controls the format of the access logs. For example:
* # The following directives define some format nicknames for use with
* # a CustomLog directive.
* 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

Some sample entries in the "common" access log are as shown:

127.0.0.1 - - [18/Oct/2009:15:41:30 +0800] "GET / HTTP/1.1" 200 44

127.0.0.1 - - [18/Oct/2009:18:36:20 +0800] "GET /t.html HTTP/1.0" 404 204

127.0.0.1 - - [18/Oct/2009:18:32:05 +0800] "get /index.html HTTP/1.0" 501 215

**5.2  Error Response**

The main role of Apache is to deliver document. When apache encounters problems and cannot meet a client's request, it generates an error code and returns an error message to explain the error. Apache provides a default set of error messages. Nonetheless, you can customize you own error response using directive ErrorDocument as follows:

* Produce a short message by providing a text string after a (").

ErrorDocument 404 "Cannot find the requested document.

* Redirect to a local page using a relative URL:
* ErrorDocument 404 /missing.html

ErrorDocument 403 //cgi-bin/forbidden.pl

* Redirect to an external page using an absolute URL. In this case, apache will send a "redirect" message to the client. The client has to issue another request to pull in the redirected page.

ErrorDocument 404 http://www.green.com/missing.html

**5.3  Directory Indexing & Listing**

If a client issues a URL selecting a directory, Apache returns a listing of that directory, if Options Indexes is on; otherwise it returns error "403 forbidden". However, if the directory contains a file called "index.html", Apache returns this "index.html" instead. You can use directive DirectoryIndex to specify the name of the indexing file. For example,

<IfModule mod\_dir.c>

# The first item takes precedence if many exist

DirectoryIndex index.html myindex.html

</IfModule>

You can control the appearance (e.g., fancy indexing) of the directory listing using directive IndexOptions (of module mod\_autoindex). See Apache documentation for more details.

To turn off automatic indexing for a directory, you can use directive "Options -indexes". Apache will return error "403 Forbidden" if a directory request is made. For example:

<Directory dir-path>

Options -Indexes

</Directory>