Apache HTTP Server Version 2.4

Apache Core Features

Description:	Core Apache HTTP Server features that are always available
Status:	Core

AcceptFilter Directive

Description:	Configures optimizations for a Protocol's Listener Sockets
Syntax:	`AcceptFilter` *`protocol accept_filter`*
Context:	server config
Status:	Core
Module:	core

This directive enables operating system specific optimizations for a listening socket by the `Protocol` type. The basic premise is for the kernel to not send a socket to the server process until either data is received or an entire HTTP Request is buffered. Only FreeBSD's Accept Filters (↗ http://www.freebsd.org/cgi/man.cgi?query=accept_filter&sektion=9) , Linux's more primitive `TCP_DEFER_ACCEPT`, and Windows' optimized AcceptEx() are currently supported.

Using `none` for an argument will disable any accept filters for that protocol. This is useful for protocols that require a server send data first, such as `ftp:` or `nntp`:

```
AcceptFilter nntp none
```

The default protocol names are `https` for port 443 and `http` for all other ports. To specify that another protocol is being used with a listening port, add the *protocol* argument to the `Listen` directive.

The default values on FreeBSD are:

```
AcceptFilter http httpready
AcceptFilter https dataready
```

The `httpready` accept filter buffers entire HTTP requests at the kernel level. Once an entire request is received, the kernel then sends it to the server. See the accf_http(9) (↗ http://www.freebsd.org/cgi/man.cgi?query=accf_http&sektion=9) man page for more details. Since HTTPS requests are encrypted, only the accf_data(9) (↗ http://www.freebsd.org/cgi/man.cgi?query=accf_data&sektion=9) filter is used.

The default values on Linux are:

```
AcceptFilter http data
AcceptFilter https data
```

Linux's `TCP_DEFER_ACCEPT` does not support buffering http requests. Any value besides `none` will enable `TCP_DEFER_ACCEPT` on that listener. For more details see the Linux tcp(7) (↗ http://man7.org/linux/man-pages/man7/tcp.7.html) man page.

The default values on Windows are:

```
AcceptFilter http connect
AcceptFilter https connect
```

Window's mpm_winnt interprets the AcceptFilter to toggle the AcceptEx() API, and does not support http protocol buffering. `connect` will use the AcceptEx() API, also retrieve the network endpoint addresses, but like `none` the `connect` option does not wait for the initial data transmission.

On Windows, `none` uses accept() rather than AcceptEx() and will not recycle sockets between connections. This is useful for network adapters with broken driver support, as well as some virtual network providers such as vpn drivers, or spam, virus or spyware filters.

The `data` AcceptFilter (Windows)

For versions 2.4.23 and prior, the Windows `data` accept filter waited until data had been transmitted and the initial data buffer and network endpoint addresses had been retrieved from the single AcceptEx() invocation. This implementation was subject to a denial of service attack and has been disabled.
Current releases of httpd default to the `connect` filter on Windows, and will fall back to `connect` if `data` is specified. Users of prior releases are encouraged to add an explicit setting of `connect` for their AcceptFilter, as shown above.

See also

- `Protocol`

AcceptPathInfo Directive

Description:	Resources accept trailing pathname information
Syntax:	`AcceptPathInfo On|Off|Default`
Default:	`AcceptPathInfo Default`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

This directive controls whether requests that contain trailing pathname information that follows an actual filename (or non-existent file in an existing directory) will be accepted or rejected. The trailing pathname information can be made available to scripts in the `PATH_INFO` environment variable.

For example, assume the location `/test/` points to a directory that contains only the single file `here.html`. Then requests for `/test/here.html/more` and `/test/nothere.html/more` both collect `/more` as `PATH_INFO`.

The three possible arguments for the `AcceptPathInfo` directive are:

`Off`

A request will only be accepted if it maps to a literal path that exists. Therefore a request with trailing pathname information after the true filename such as `/test/here.html/more` in the above example will return a 404 NOT FOUND error.

`On`
A request will be accepted if a leading path component maps to a file that exists. The above example `/test/here.html/more` will be accepted if `/test/here.html` maps to a valid file.

`Default`
The treatment of requests with trailing pathname information is determined by the handler responsible for the request. The core handler for normal files defaults to rejecting `PATH_INFO` requests. Handlers that serve scripts, such as cgi-script and isapi-handler, generally accept `PATH_INFO` by default.

The primary purpose of the `AcceptPathInfo` directive is to allow you to override the handler's choice of accepting or rejecting `PATH_INFO`. This override is required, for example, when you use a filter (↗ ../filter.html) , such as INCLUDES (↗ mod_include.html) , to generate content based on `PATH_INFO`. The core handler would usually reject the request, so you can use the following configuration to enable such a script:

```
<Files "mypaths.shtml">
  Options +Includes
  SetOutputFilter INCLUDES
  AcceptPathInfo On
</Files>
```

AccessFileName Directive

Description:	Name of the distributed configuration file
Syntax:	`AccessFileName filename [filename] ...`
Default:	`AccessFileName .htaccess`
Context:	server config, virtual host
Status:	Core
Module:	core

While processing a request, the server looks for the first existing configuration file from this list of names in every directory of the path to the document, if distributed configuration files are enabled for that directory (↗ #allowoverride) . For example:

```
AccessFileName .acl
```

Before returning the document `/usr/local/web/index.html`, the server will read `/.acl`, `/usr/.acl`, `/usr/local/.acl` and `/usr/local/web/.acl` for directives unless they have been disabled with:

```
<Directory "/">
    AllowOverride None
</Directory>
```

See also

- `AllowOverride`
- Configuration Files
- .htaccess Files

AddDefaultCharset Directive

Description:	Default charset parameter to be added when a response content-type is `text/plain` or `text/html`
Syntax:	`AddDefaultCharset On|Off|charset`
Default:	`AddDefaultCharset Off`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

This directive specifies a default value for the media type charset parameter (the name of a character encoding) to be added to a response if and only if the response's content-type is either `text/plain` or `text/html`. This should override any charset specified in the body of the response via a `META` element, though the exact behavior is often dependent on the user's client configuration. A setting of `AddDefaultCharset Off` disables this functionality. `AddDefaultCharset On` enables a default charset of `iso-8859-1`. Any other value is assumed to be the *charset* to be used, which should be one of the IANA registered charset values (↗ http://www.iana.org/assignments/character-sets) for use in Internet media types (MIME types). For example:

```
AddDefaultCharset utf-8
```

`AddDefaultCharset` should only be used when all of the text resources to which it applies are known to be in that character encoding and it is too inconvenient to label their charset individually. One such example is to add the charset parameter to resources containing generated content, such as legacy CGI scripts, that might be vulnerable to cross-site scripting attacks due to user-provided data being included in the output. Note, however, that a better solution is to just fix (or delete) those scripts, since setting a default charset does not protect users that have enabled the "auto-detect character encoding" feature on their browser.

See also

- `AddCharset`

AllowEncodedSlashes Directive

Description:	Determines whether encoded path separators in URLs are allowed to be passed through
Syntax:	`AllowEncodedSlashes On|Off|NoDecode`
Default:	`AllowEncodedSlashes Off`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	NoDecode option available in 2.3.12 and later.

The AllowEncodedSlashes directive allows URLs which contain encoded path separators (%2F for / and additionally %5C for \ on accordant systems) to be used in the path info.

With the default value, Off, such URLs are refused with a 404 (Not found) error.

With the value On, such URLs are accepted, and encoded slashes are decoded like all other encoded characters.

With the value NoDecode, such URLs are accepted, but encoded slashes are not decoded but left in their encoded state.

Turning AllowEncodedSlashes On is mostly useful when used in conjunction with PATH_INFO.

Note

If encoded slashes are needed in path info, use of NoDecode is strongly recommended as a security measure. Allowing slashes to be decoded could potentially allow unsafe paths.

See also

- AcceptPathInfo

AllowOverride Directive

Description:	Types of directives that are allowed in .htaccess files
Syntax:	AllowOverride All|None|*directive-type* [*directive-type*] ...
Default:	AllowOverride None (2.3.9 and later), AllowOverride All (2.3.8 and earlier)
Context:	directory
Status:	Core
Module:	core

When the server finds an .htaccess file (as specified by AccessFileName), it needs to know which directives declared in that file can override earlier configuration directives.

Only available in <Directory> sections

AllowOverride is valid only in <Directory> sections specified without regular expressions, not in <Location>, <DirectoryMatch> or <Files> sections.

When this directive is set to None and AllowOverrideList is set to None, .htaccess (↗ #accessfilename) files are completely ignored. In this case, the server will not even attempt to read .htaccess files in the filesystem.

When this directive is set to All, then any directive which has the .htaccess Context (↗ directive-dict.html#Context) is allowed in .htaccess files.

The *directive-type* can be one of the following groupings of directives. (See the override class index (↗ overrides.html) for an up-to-date listing of which directives are enabled by each *directive-type*.)

AuthConfig
Allow use of the authorization directives (AuthDBMGroupFile, AuthDBMUserFile, AuthGroupFile, AuthName, AuthType, AuthUserFile, Require, *etc.*).

FileInfo
Allow use of the directives controlling document types (ErrorDocument, ForceType, LanguagePriority, SetHandler, SetInputFilter, SetOutputFilter, and mod_mime Add* and Remove* directives), document meta data (Header, RequestHeader, SetEnvIf, SetEnvIfNoCase, BrowserMatch, CookieExpires, CookieDomain, CookieStyle, CookieTracking, CookieName), mod_rewrite directives (RewriteEngine, RewriteOptions, RewriteBase, RewriteCond, RewriteRule), mod_alias directives (Redirect, RedirectTemp, RedirectPermanent, RedirectMatch), and Action from mod_actions.

Indexes
Allow use of the directives controlling directory indexing (AddDescription, AddIcon, AddIconByEncoding, AddIconByType, DefaultIcon, DirectoryIndex, FancyIndexing, HeaderName, IndexIgnore, IndexOptions, ReadmeName, *etc.*).

Limit
Allow use of the directives controlling host access (Allow, Deny and Order).

Nonfatal=[Override|Unknown|All]
Allow use of AllowOverride option to treat syntax errors in .htaccess as nonfatal. Instead of causing an Internal Server Error, disallowed or unrecognised directives will be ignored and a warning logged:

- **Nonfatal=Override** treats directives forbidden by AllowOverride as nonfatal.
- **Nonfatal=Unknown** treats unknown directives as nonfatal. This covers typos and directives implemented by a module that's not present.
- **Nonfatal=All** treats both the above as nonfatal.

Note that a syntax error in a valid directive will still cause an internal server error.

Security

Nonfatal errors may have security implications for .htaccess users. For example, if AllowOverride disallows AuthConfig, users' configuration designed to restrict access to a site will be disabled.

Options[=*Option*,...]
Allow use of the directives controlling specific directory features (Options and XBitHack). An equal sign may be given followed by a comma-separated list, without spaces, of options that may be set using the Options command.

Implicit disabling of Options

Even though the list of options that may be used in .htaccess files can be limited with this directive, as long as any Options directive is allowed any other inherited option can be disabled by using the non-relative syntax. In other words, this mechanism cannot force a specific option to remain *set* while allowing any others to be set.

```
AllowOverride Options=Indexes,MultiViews
```

Example:

```
AllowOverride AuthConfig Indexes
```

In the example above, all directives that are neither in the group `AuthConfig` nor `Indexes` cause an internal server error.

> For security and performance reasons, do not set `AllowOverride` to anything other than `None` in your `<Directory "/">` block. Instead, find (or create) the `<Directory>` block that refers to the directory where you're actually planning to place a `.htaccess` file.

See also

- `AccessFileName`
- `AllowOverrideList`
- Configuration Files
- .htaccess Files
- Override Class Index for .htaccess

AllowOverrideList Directive

Description:	Individual directives that are allowed in `.htaccess` files
Syntax:	`AllowOverrideList None|`*`directive`* `[`*`directive-type`*`] ...`
Default:	`AllowOverrideList None`
Context:	directory
Status:	Core
Module:	core

When the server finds an `.htaccess` file (as specified by `AccessFileName`), it needs to know which directives declared in that file can override earlier configuration directives.

> **Only available in <Directory> sections**
>
> `AllowOverrideList` is valid only in `<Directory>` sections specified without regular expressions, not in `<Location>`, `<DirectoryMatch>` or `<Files>` sections.

When this directive is set to `None` and `AllowOverride` is set to `None`, then .htaccess (↗ #accessfilename) files are completely ignored. In this case, the server will not even attempt to read `.htaccess` files in the filesystem.

Example:

```
AllowOverride None
AllowOverrideList Redirect RedirectMatch
```

In the example above, only the `Redirect` and `RedirectMatch` directives are allowed. All others will cause an internal server error.

Example:

```
AllowOverride AuthConfig
AllowOverrideList CookieTracking CookieName
```

In the example above, `AllowOverride` grants permission to the `AuthConfig` directive grouping and `AllowOverrideList` grants permission to only two directives from the `FileInfo` directive grouping. All others will cause an internal server error.

See also

- `AccessFileName`
- `AllowOverride`
- Configuration Files
- .htaccess Files

CGIMapExtension Directive

Description:	Technique for locating the interpreter for CGI scripts
Syntax:	`CGIMapExtension` *`cgi-path .extension`*
Context:	directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	NetWare only

This directive is used to control how Apache httpd finds the interpreter used to run CGI scripts. For example, setting `CGIMapExtension sys:\foo.nlm .foo` will cause all CGI script files with a `.foo` extension to be passed to the FOO interpreter.

CGIPassAuth Directive

Description:	Enables passing HTTP authorization headers to scripts as CGI variables
Syntax:	`CGIPassAuth On|Off`
Default:	`CGIPassAuth Off`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Core
Module:	core
Compatibility:	Available in Apache HTTP Server 2.4.13 and later

`CGIPassAuth` allows scripts access to HTTP authorization headers such as `Authorization`, which is required for scripts that implement HTTP Basic authentication. Normally these HTTP headers are hidden from scripts. This is to disallow scripts from seeing user ids and passwords used to access the server when HTTP Basic authentication is enabled in the web server. This directive should be used when scripts are allowed to implement HTTP Basic authentication.

This directive can be used instead of the compile-time setting `SECURITY_HOLE_PASS_AUTHORIZATION` which has been available in previous versions of Apache HTTP Server.

The setting is respected by any modules which use `ap_add_common_vars()`, such as `mod_cgi`, `mod_cgid`, `mod_proxy_fcgi`, `mod_proxy_scgi`, and so on. Notably, it affects modules which don't handle the request in the usual sense but still use this API; examples of this are `mod_include` and `mod_ext_filter`. Third-party modules that don't use `ap_add_common_vars()` may choose to respect the setting as well.

CGIVar Directive

Description:	Controls how some CGI variables are set
Syntax:	`CGIVar` *`variable rule`*
Context:	directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Available in Apache HTTP Server 2.4.21 and later

This directive controls how some CGI variables are set.

REQUEST_URI rules:

`original-uri` **(default)**
: The value is taken from the original request line, and will not reflect internal redirects or subrequests which change the requested resource.

`current-uri`
: The value reflects the resource currently being processed, which may be different than the original request from the client due to internal redirects or subrequests.

ContentDigest Directive

Description:	Enables the generation of `Content-MD5` HTTP Response headers
Syntax:	`ContentDigest On|Off`
Default:	`ContentDigest Off`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Core
Module:	core

This directive enables the generation of `Content-MD5` headers as defined in RFC1864 respectively RFC2616.

MD5 is an algorithm for computing a "message digest" (sometimes called "fingerprint") of arbitrary-length data, with a high degree of confidence that any alterations in the data will be reflected in alterations in the message digest.

The `Content-MD5` header provides an end-to-end message integrity check (MIC) of the entity-body. A proxy or client may check this header for detecting accidental modification of the entity-body in transit. Example header:

```
Content-MD5: AuLb7Dp1rqtRtxz2m9kRpA==
```

Note that this can cause performance problems on your server since the message digest is computed on every request (the values are not cached).

`Content-MD5` is only sent for documents served by the `core`, and not by any module. For example, SSI documents, output from CGI scripts, and byte range responses do not have this header.

DefaultRuntimeDir Directive

Description:	Base directory for the server run-time files
Syntax:	`DefaultRuntimeDir` *`directory-path`*
Default:	`DefaultRuntimeDir DEFAULT_REL_RUNTIMEDIR (logs/)`
Context:	server config
Status:	Core
Module:	core
Compatibility:	Available in Apache 2.4.2 and later

The `DefaultRuntimeDir` directive sets the directory in which the server will create various run-time files (shared memory, locks, etc.). If set as a relative path, the full path will be relative to `ServerRoot`.

Example

```
DefaultRuntimeDir scratch/
```

The default location of `DefaultRuntimeDir` may be modified by changing the `DEFAULT_REL_RUNTIMEDIR` #define at build time.

Note: `ServerRoot` should be specified before this directive is used. Otherwise, the default value of `ServerRoot` would be used to set the base directory.

See also

- the security tips for information on how to properly set permissions on the `ServerRoot`

DefaultType Directive

Description:	This directive has no effect other than to emit warnings if the value is not `none`. In prior versions, DefaultType would specify a default media type to assign to response content for which no other media type configuration could be found.
Syntax:	`DefaultType` *`media-type|none`*
Default:	`DefaultType none`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	The argument `none` is available in Apache httpd 2.2.7 and later. All other choices are DISABLED for 2.3.x and later.

This directive has been disabled. For backwards compatibility of configuration files, it may be specified with the value `none`, meaning no default media type. For example:

```
DefaultType None
```

`DefaultType None` is only available in httpd-2.2.7 and later.

Use the mime.types configuration file and the `AddType` to configure media type assignments via file extensions, or the `ForceType` directive to configure the media type for specific resources. Otherwise, the server will send the response without a Content-Type header field and the recipient may attempt to guess the media type.

Define Directive

Description:	Define a variable
Syntax:	`Define` *`parameter-name`* `[`*`parameter-value`*`]`
Context:	server config, virtual host, directory
Status:	Core
Module:	core

In its one parameter form, `Define` is equivalent to passing the `-D` argument to `httpd`. It can be used to toggle the use of `<IfDefine>` sections without needing to alter `-D` arguments in any startup scripts.

In addition to that, if the second parameter is given, a config variable is set to this value. The variable can be used in the configuration using the `${VAR}` syntax. The variable is always globally defined and not limited to the scope of the surrounding config section.

```
<IfDefine TEST>
  Define servername test.example.com
</IfDefine>
<IfDefine !TEST>
  Define servername www.example.com
  Define SSL
</IfDefine>

DocumentRoot "/var/www/${servername}/htdocs"
```

Variable names may not contain colon ":" characters, to avoid clashes with `RewriteMap`'s syntax.

Virtual Host scope and pitfalls

While this directive is supported in virtual host context, the changes it makes are visible to any later configuration directives, beyond any enclosing virtual host.

See also

- `UnDefine`
- `IfDefine`

<Directory> Directive

Description:	Enclose a group of directives that apply only to the named file-system directory, sub-directories, and their contents.
Syntax:	`<Directory` *`directory-path`*`> ... </Directory>`
Context:	server config, virtual host
Status:	Core
Module:	core

`<Directory>` and `</Directory>` are used to enclose a group of directives that will apply only to the named directory, sub-directories of that directory, and the files within the respective directories. Any directive that is allowed in a directory context may be used. *Directory-path* is either the full path to a directory, or a wild-card string using Unix shell-style matching. In a wild-card string, `?` matches any single character, and `*` matches any sequences of characters. You may also use `[]` character ranges. None of the wildcards match a `/' character, so `<Directory "/*/public_html">` will not match `/home/user/public_html`, but `<Directory "/home/*/public_html">` will match. Example:

```
<Directory "/usr/local/httpd/htdocs">
  Options Indexes FollowSymLinks
</Directory>
```

Directory paths *may* be quoted, if you like, however, it *must* be quoted if the path contains spaces. This is because a space would otherwise indicate the end of an argument.

Be careful with the *directory-path* arguments: They have to literally match the filesystem path which Apache httpd uses to access the files. Directives applied to a particular `<Directory>` will not apply to files accessed from that same directory via a different path, such as via different symbolic links.

Regular expressions (↗ ../glossary.html#regex) can also be used, with the addition of the ~ character. For example:

```
<Directory ~ "^/www/[0-9]{3}">

</Directory>
```

would match directories in `/www/` that consisted of three numbers.

If multiple (non-regular expression) `<Directory>` sections match the directory (or one of its parents) containing a document, then the directives are applied in the order of shortest match first, interspersed with the directives from the .htaccess (↗ #accessfilename) files. For example, with

```
<Directory "/">
  AllowOverride None
</Directory>

<Directory "/home">
```

```
  AllowOverride FileInfo
</Directory>
```

for access to the document `/home/web/dir/doc.html` the steps are:

- Apply directive `AllowOverride None` (disabling `.htaccess` files).
- Apply directive `AllowOverride FileInfo` (for directory `/home`).
- Apply any `FileInfo` directives in `/home/.htaccess`, `/home/web/.htaccess` and `/home/web/dir/.htaccess` in that order.

Regular expressions are not considered until after all of the normal sections have been applied. Then all of the regular expressions are tested in the order they appeared in the configuration file. For example, with

```
<Directory ~ "abc$">
  # ... directives here ...
</Directory>
```

the regular expression section won't be considered until after all normal `<Directory>`s and `.htaccess` files have been applied. Then the regular expression will match on `/home/abc/public_html/abc` and the corresponding `<Directory>` will be applied.

Note that the default access for `<Directory "/">` is to permit all access. This means that Apache httpd will serve any file mapped from an URL. It is recommended that you change this with a block such as

```
<Directory "/">
  Require all denied
</Directory>
```

and then override this for directories you *want* accessible. See the Security Tips page for more details.

The directory sections occur in the `httpd.conf` file. `<Directory>` directives cannot nest, and cannot appear in a `<Limit>` or `<LimitExcept>` section.

See also

- How <Directory>, <Location> and <Files> sections work for an explanation of how these different sections are combined when a request is received

<DirectoryMatch> Directive

Description:	Enclose directives that apply to the contents of file-system directories matching a regular expression.
Syntax:	`<DirectoryMatch *regex*> ... </DirectoryMatch>`
Context:	server config, virtual host
Status:	Core
Module:	core

`<DirectoryMatch>` and `</DirectoryMatch>` are used to enclose a group of directives which will apply only to the named directory (and the files within), the same as `<Directory>`. However, it takes as an argument a regular expression (↗ ../glossary.html#regex) . For example:

```
<DirectoryMatch "^/www/(.+/)?[0-9]{3}/">
    # ...
</DirectoryMatch>
```

matches directories in `/www/` (or any subdirectory thereof) that consist of three numbers.

Compatibility

Prior to 2.3.9, this directive implicitly applied to sub-directories (like `<Directory>`) and could not match the end of line symbol ($). In 2.3.9 and later, only directories that match the expression are affected by the enclosed directives.

Trailing Slash

This directive applies to requests for directories that may or may not end in a trailing slash, so expressions that are anchored to the end of line ($) must be written with care.

From 2.4.8 onwards, named groups and backreferences are captured and written to the environment with the corresponding name prefixed with "MATCH_" and in upper case. This allows elements of paths to be referenced from within expressions (↗ ../expr.html) and modules like `mod_rewrite`. In order to prevent confusion, numbered (unnamed) backreferences are ignored. Use named groups instead.

```
<DirectoryMatch "^/var/www/combined/(?<sitename>[^/]+)">
    Require ldap-group cn=%{env:MATCH_SITENAME},ou=combined,o=Example
</DirectoryMatch>
```

See also

- `<Directory>` for a description of how regular expressions are mixed in with normal `<Directory>`s
- How <Directory>, <Location> and <Files> sections work for an explanation of how these different sections are combined when a request is received

DocumentRoot Directive

Description:	Directory that forms the main document tree visible from the web
Syntax:	`DocumentRoot *directory-path*`
Default:	`DocumentRoot "/usr/local/apache/htdocs"`
Context:	server config, virtual host
Status:	Core
Module:	core

This directive sets the directory from which `httpd` will serve files. Unless matched by a directive like `Alias`, the server appends the path from the requested URL to the document root to make the path to the document. Example:

```
DocumentRoot "/usr/web"
```

then an access to `http://my.example.com/index.html` refers to `/usr/web/index.html`. If the *directory-path* is not absolute then it is assumed to be relative to the `ServerRoot`.

The `DocumentRoot` should be specified without a trailing slash.

See also

- Mapping URLs to Filesystem Locations

<Else> Directive

Description:	Contains directives that apply only if the condition of a previous `<If>` or `<ElseIf>` section is not satisfied by a request at runtime
Syntax:	`<Else> ... </Else>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core
Compatibility:	Nested conditions are evaluated in 2.4.26 and later

The `<Else>` applies the enclosed directives if and only if the most recent `<If>` or `<ElseIf>` section in the same scope has not been applied. For example: In

```
<If "-z req('Host')">
  # ...
</If>
<Else>
  # ...
</Else>
```

The `<If>` would match HTTP/1.0 requests without a *Host:* header and the `<Else>` would match requests with a *Host:* header.

See also

- `<If>`
- `<ElseIf>`
- How <Directory>, <Location>, <Files> sections work for an explanation of how these different sections are combined when a request is received. `<If>`, `<ElseIf>`, and `<Else>` are applied last.

<ElseIf> Directive

Description:	Contains directives that apply only if a condition is satisfied by a request at runtime while the condition of a previous `<If>` or `<ElseIf>` section is not satisfied
Syntax:	`<ElseIf *expression*> ... </ElseIf>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core
Compatibility:	Nested conditions are evaluated in 2.4.26 and later

The `<ElseIf>` applies the enclosed directives if and only if both the given condition evaluates to true and the most recent `<If>` or `<ElseIf>` section in the same scope has not been applied. For example: In

```
<If "-R '10.1.0.0/16'">
  #...
</If>
<ElseIf "-R '10.0.0.0/8'">
  #...
</ElseIf>
<Else>
  #...
</Else>
```

The `<ElseIf>` would match if the remote address of a request belongs to the subnet 10.0.0.0/8 but not to the subnet 10.1.0.0/16.

See also

- Expressions in Apache HTTP Server, for a complete reference and more examples.
- `<If>`
- `<Else>`
- How <Directory>, <Location>, <Files> sections work for an explanation of how these different sections are combined when a request is received. `<If>`, `<ElseIf>`, and `<Else>` are applied last.

EnableMMAP Directive

Description:	Use memory-mapping to read files during delivery
Syntax:	`EnableMMAP On|Off`
Default:	`EnableMMAP On`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

This directive controls whether the `httpd` may use memory-mapping if it needs to read the contents of a file during delivery. By default, when the handling of a request requires access to the data within a file -- for example, when delivering a server-parsed file using `mod_include` -- Apache httpd memory-maps the file if the OS supports it.

This memory-mapping sometimes yields a performance improvement. But in some environments, it is better to disable the memory-mapping to prevent operational problems:

- On some multiprocessor systems, memory-mapping can reduce the performance of the `httpd`.
- Deleting or truncating a file while `httpd` has it memory-mapped can cause `httpd` to crash with a segmentation fault.

For server configurations that are vulnerable to these problems, you should disable memory-mapping of delivered files by specifying:

```
EnableMMAP Off
```

For NFS mounted files, this feature may be disabled explicitly for the offending files by specifying:

```
<Directory "/path-to-nfs-files">
  EnableMMAP Off
</Directory>
```

EnableSendfile Directive

Description:	Use the kernel sendfile support to deliver files to the client
Syntax:	`EnableSendfile On|Off`
Default:	`EnableSendfile Off`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Default changed to Off in version 2.3.9.

This directive controls whether `httpd` may use the sendfile support from the kernel to transmit file contents to the client. By default, when the handling of a request requires no access to the data within a file -- for example, when delivering a static file -- Apache httpd uses sendfile to deliver the file contents without ever reading the file if the OS supports it.

This sendfile mechanism avoids separate read and send operations, and buffer allocations. But on some platforms or within some filesystems, it is better to disable this feature to avoid operational problems:

- Some platforms may have broken sendfile support that the build system did not detect, especially if the binaries were built on another box and moved to such a machine with broken sendfile support.
- On Linux the use of sendfile triggers TCP-checksum offloading bugs on certain networking cards when using IPv6.
- On Linux on Itanium, `sendfile` may be unable to handle files over 2GB in size.
- With a network-mounted `DocumentRoot` (e.g., NFS, SMB, CIFS, FUSE), the kernel may be unable to serve the network file through its own cache.

For server configurations that are not vulnerable to these problems, you may enable this feature by specifying:

```
EnableSendfile On
```

For network mounted files, this feature may be disabled explicitly for the offending files by specifying:

```
<Directory "/path-to-nfs-files">
  EnableSendfile Off
</Directory>
```

Please note that the per-directory and .htaccess configuration of `EnableSendfile` is not supported by `mod_cache_disk`. Only global definition of `EnableSendfile` is taken into account by the module.

Error Directive

Description:	Abort configuration parsing with a custom error message
Syntax:	`Error` *`message`*
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core
Compatibility:	2.3.9 and later

If an error can be detected within the configuration, this directive can be used to generate a custom error message, and halt configuration parsing. The typical use is for reporting required modules which are missing from the configuration.

```
# Example
# ensure that mod_include is loaded
<IfModule !include_module>
  Error "mod_include is required by mod_foo.  Load it with LoadModule."
</IfModule>

# ensure that exactly one of SSL,NOSSL is defined
<IfDefine SSL>
<IfDefine NOSSL>
  Error "Both SSL and NOSSL are defined.  Define only one of them."
</IfDefine>
</IfDefine>
<IfDefine !SSL>
<IfDefine !NOSSL>
  Error "Either SSL or NOSSL must be defined."
</IfDefine>
</IfDefine>
```

Note

This directive is evaluated and configuration processing time, not at runtime. As a result, this directive cannot be conditonally evaluated by enclosing it in an `<If>` section.

ErrorDocument Directive

Description:	What the server will return to the client in case of an error
Syntax:	`ErrorDocument error-code document`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

In the event of a problem or error, Apache httpd can be configured to do one of four things,

1. output a simple hardcoded error message
2. output a customized message
3. internally redirect to a local *URL-path* to handle the problem/error
4. redirect to an external *URL* to handle the problem/error

The first option is the default, while options 2-4 are configured using the `ErrorDocument` directive, which is followed by the HTTP response code and a URL or a message. Apache httpd will sometimes offer additional information regarding the problem/error.

From 2.4.13, expression syntax (↗ ../expr.html) can be used inside the directive to produce dynamic strings and URLs.

URLs can begin with a slash (/) for local web-paths (relative to the `DocumentRoot`), or be a full URL which the client can resolve. Alternatively, a message can be provided to be displayed by the browser. Note that deciding whether the parameter is an URL, a path or a message is performed before any expression is parsed. Examples:

```
ErrorDocument 500 http://example.com/cgi-bin/server-error.cgi
ErrorDocument 404 /errors/bad_urls.php
ErrorDocument 401 /subscription_info.html
ErrorDocument 403 "Sorry, can't allow you access today"
ErrorDocument 403 Forbidden!
ErrorDocument 403 /errors/forbidden.py?referrer=%{escape:%{HTTP_REFERER}}
```

Additionally, the special value `default` can be used to specify Apache httpd's simple hardcoded message. While not required under normal circumstances, `default` will restore Apache httpd's simple hardcoded message for configurations that would otherwise inherit an existing `ErrorDocument`.

```
ErrorDocument 404 /cgi-bin/bad_urls.pl

<Directory "/web/docs">
  ErrorDocument 404 default
</Directory>
```

Note that when you specify an `ErrorDocument` that points to a remote URL (ie. anything with a method such as `http` in front of it), Apache HTTP Server will send a redirect to the client to tell it where to find the document, even if the document ends up being on the same server. This has several implications, the most important being that the client will not receive the original error status code, but instead will receive a redirect status code. This in turn can confuse web robots and other clients which try to determine if a URL is valid using the status code. In addition, if you use a remote URL in an `ErrorDocument 401`, the client will not know to prompt the user for a password since it will not receive the 401 status code. Therefore, **if you use an `ErrorDocument 401` directive, then it must refer to a local document.**

Microsoft Internet Explorer (MSIE) will by default ignore server-generated error messages when they are "too small" and substitute its own "friendly" error messages. The size threshold varies depending on the type of error, but in general, if you make your error document greater than 512 bytes, then MSIE will show the server-generated error rather than masking it. More information is available in Microsoft Knowledge Base article Q294807 (↗ http://support.microsoft.com/default.aspx?scid=kb;en-us;Q294807) .

Although most error messages can be overridden, there are certain circumstances where the internal messages are used regardless of the setting of `ErrorDocument`. In particular, if a malformed request is detected, normal request processing will be immediately halted and the internal error message returned. This is necessary to guard against security problems caused by bad requests.

If you are using mod_proxy, you may wish to enable `ProxyErrorOverride` so that you can provide custom error messages on behalf of your Origin servers. If you don't enable ProxyErrorOverride, Apache httpd will not generate custom error documents for proxied content.

See also

- documentation of customizable responses

ErrorLog Directive

Description:	Location where the server will log errors
Syntax:	`ErrorLog file-path|syslog[:[facility][:tag]]`
Default:	`ErrorLog logs/error_log (Unix) ErrorLog logs/error.log (Windows and OS/2)`
Context:	server config, virtual host
Status:	Core
Module:	core

The `ErrorLog` directive sets the name of the file to which the server will log any errors it encounters. If the *file-path* is not absolute then it is assumed to be relative to the `ServerRoot`.

```
ErrorLog "/var/log/httpd/error_log"
```

If the *file-path* begins with a pipe character "`|`" then it is assumed to be a command to spawn to handle the error log.

```
ErrorLog "|/usr/local/bin/httpd_errors"
```

See the notes on piped logs (↗ ../logs.html#piped) for more information.

Using `syslog` instead of a filename enables logging via syslogd(8) if the system supports it. The default is to use syslog facility `local7`, but you can override this by using the `syslog:facility` syntax where *facility* can be one of the names usually documented in syslog(1). The facility is effectively global, and if it is changed in individual virtual hosts, the final facility specified affects the entire server. Same rules apply for the syslog tag, which by default uses the Apache binary name, `httpd` in most cases. You can also override this by using the `syslog::tag` syntax.

```
ErrorLog syslog:user
ErrorLog syslog:user:httpd.srv1
ErrorLog syslog::httpd.srv2
```

SECURITY: See the security tips (↗ ../misc/security_tips.html#serverroot) document for details on why your security could be compromised if the directory where log files are stored is writable by anyone other than the user that starts the server.

Note

When entering a file path on non-Unix platforms, care should be taken to make sure that only forward slashes are used even though the platform may allow the use of back slashes. In general it is a good idea to always use forward slashes throughout the configuration files.

See also

- `LogLevel`
- Apache HTTP Server Log Files

ErrorLogFormat Directive

Description:	Format specification for error log entries
Syntax:	`ErrorLogFormat [connection|request] *format*`
Context:	server config, virtual host
Status:	Core
Module:	core

`ErrorLogFormat` allows to specify what supplementary information is logged in the error log in addition to the actual log message.

```
#Simple example
ErrorLogFormat "[%t] [%l] [pid %P] %F: %E: [client %a] %M"
```

Specifying `connection` or `request` as first parameter allows to specify additional formats, causing additional information to be logged when the first message is logged for a specific connection or request, respectively. This additional information is only logged once per connection/request. If a connection or request is processed without causing any log message, the additional information is not logged either.

It can happen that some format string items do not produce output. For example, the Referer header is only present if the log message is associated to a request and the log message happens at a time when the Referer header has already been read from the client. If no output is produced, the default behavior is to delete everything from the preceding space character to the next space character. This means the log line is implicitly divided into fields on non-whitespace to whitespace transitions. If a format string item does not produce output, the whole field is omitted. For example, if the remote address `%a` in the log format `[%t] [%l] [%a] %M` is not available, the surrounding brackets are not logged either. Space characters can be escaped with a backslash to prevent them from delimiting a field. The combination '% ' (percent space) is a zero-width field delimiter that does not produce any output.

The above behavior can be changed by adding modifiers to the format string item. A `-` (minus) modifier causes a minus to be logged if the respective item does not produce any output. In once-per-connection/request formats, it is also possible to use the `+` (plus) modifier. If an item with the plus modifier does not produce any output, the whole line is omitted.

A number as modifier can be used to assign a log severity level to a format item. The item will only be logged if the severity of the log message is not higher than the specified log severity level. The number can range from 1 (alert) over 4 (warn) and 7 (debug) to 15 (trace8).

For example, here's what would happen if you added modifiers to the `%{Referer}i` token, which logs the `Referer` request header.

Modified Token	Meaning
`%-{Referer}i`	Logs a `-` if `Referer` is not set.
`%+{Referer}i`	Omits the entire line if `Referer` is not set.
`%4{Referer}i`	Logs the `Referer` only if the log message severity is higher than 4.

Some format string items accept additional parameters in braces.

Format String	Description
`%%`	The percent sign
`%a`	Client IP address and port of the request
`%{c}a`	Underlying peer IP address and port of the connection (see the `mod_remoteip` module)
`%A`	Local IP-address and port
`%{*name*}e`	Request environment variable *name*
`%E`	APR/OS error status code and string
`%F`	Source file name and line number of the log call
`%{*name*}i`	Request header *name*
`%k`	Number of keep-alive requests on this connection
`%l`	Loglevel of the message
`%L`	Log ID of the request
`%{c}L`	Log ID of the connection
`%{C}L`	Log ID of the connection if used in connection scope, empty otherwise
`%m`	Name of the module logging the message
`%M`	The actual log message
`%{*name*}n`	Request note *name*
`%P`	Process ID of current process
`%T`	Thread ID of current thread
`%{g}T`	System unique thread ID of current thread (the same ID as displayed by e.g. `top`; currently Linux only)
`%t`	The current time
`%{u}t`	The current time including micro-seconds
`%{cu}t`	The current time in compact ISO 8601 format, including micro-seconds
`%v`	The canonical `ServerName` of the current server.
`%V`	The server name of the server serving the request according to the `UseCanonicalName` setting.
`\ ` (backslash space)	Non-field delimiting space

`% ` (percent space)	Field delimiter (no output)

The log ID format `%L` produces a unique id for a connection or request. This can be used to correlate which log lines belong to the same connection or request, which request happens on which connection. A `%L` format string is also available in `mod_log_config` to allow to correlate access log entries with error log lines. If `mod_unique_id` is loaded, its unique id will be used as log ID for requests.

```
#Example (default format for threaded MPMs)
ErrorLogFormat "[%{u}t] [%-m:%l] [pid %P:tid %T] %7F: %E: [client\ %a] %M% ,\ referer\ %{Referer}i"
```

This would result in error messages such as:

```
[Thu May 12 08:28:57.652118 2011] [core:error] [pid 8777:tid 4326490112] [client ::1:58619] File does not exist:
/usr/local/apache2/htdocs/favicon.ico
```

Notice that, as discussed above, some fields are omitted entirely because they are not defined.

```
#Example (similar to the 2.2.x format)
ErrorLogFormat "[%t] [%l] %7F: %E: [client\ %a] %M% ,\ referer\ %{Referer}i"
```

```
#Advanced example with request/connection log IDs
ErrorLogFormat "[%{uc}t] [%-m:%-l] [R:%L] [C:%{C}L] %7F: %E: %M"
ErrorLogFormat request "[%{uc}t] [R:%L] Request %k on C:%{c}L pid:%P tid:%T"
ErrorLogFormat request "[%{uc}t] [R:%L] UA:'%+{User-Agent}i'"
ErrorLogFormat request "[%{uc}t] [R:%L] Referer:'%+{Referer}i'"
ErrorLogFormat connection "[%{uc}t] [C:%{c}L] remote\ %a local\ %A"
```

See also

- `ErrorLog`
- `LogLevel`
- Apache HTTP Server Log Files

ExtendedStatus Directive

Description:	Keep track of extended status information for each request
Syntax:	`ExtendedStatus On|Off`
Default:	`ExtendedStatus Off[*]`
Context:	server config
Status:	Core
Module:	core

This option tracks additional data per worker about the currently executing request and creates a utilization summary. You can see these variables during runtime by configuring `mod_status`. Note that other modules may rely on this scoreboard.

This setting applies to the entire server and cannot be enabled or disabled on a virtualhost-by-virtualhost basis. The collection of extended status information can slow down the server. Also note that this setting cannot be changed during a graceful restart.

> Note that loading `mod_status` will change the default behavior to ExtendedStatus On, while other third party modules may do the same. Such modules rely on collecting detailed information about the state of all workers. The default is changed by `mod_status` beginning with version 2.3.6. The previous default was always Off.

FileETag Directive

Description:	File attributes used to create the ETag HTTP response header for static files
Syntax:	`FileETag` *`component`* `...`
Default:	`FileETag MTime Size`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	The default used to be "INode MTime Size" in 2.3.14 and earlier.

The `FileETag` directive configures the file attributes that are used to create the `ETag` (entity tag) response header field when the document is based on a static file. (The `ETag` value is used in cache management to save network bandwidth.) The `FileETag` directive allows you to choose which of these -- if any -- should be used. The recognized keywords are:

INode
    The file's i-node number will be included in the calculation

MTime
    The date and time the file was last modified will be included

Size
    The number of bytes in the file will be included

All
    All available fields will be used. This is equivalent to:

```
FileETag INode MTime Size
```

Digest
    If a document is file-based, the `ETag` field will be calculated by taking the digest over the file.

None
    If a document is file-based, no `ETag` field will be included in the response

The `INode`, `MTime`, `Size` and `Digest` keywords may be prefixed with either `+` or `-`, which allow changes to be made to the default setting inherited from a broader scope. Any keyword appearing without such a prefix immediately and completely cancels the inherited setting.

If a directory's configuration includes `FileETag INode MTime Size`, and a subdirectory's includes `FileETag -INode`, the setting for that subdirectory (which will be inherited by any sub-subdirectories that don't override it) will be equivalent to `FileETag MTime Size`.

> **Server Side Includes**
>
> An ETag is not generated for responses parsed by `mod_include` since the response entity can change without a change of the INode, MTime, Size or Digest of the static file with embedded SSI directives.

<Files> Directive

Description:	Contains directives that apply to matched filenames
Syntax:	`<Files filename> ... </Files>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `<Files>` directive limits the scope of the enclosed directives by filename. It is comparable to the `<Directory>` and `<Location>` directives. It should be matched with a `</Files>` directive. The directives given within this section will be applied to any object with a basename (last component of filename) matching the specified filename. `<Files>` sections are processed in the order they appear in the configuration file, after the `<Directory>` sections and `.htaccess` files are read, but before `<Location>` sections. Note that `<Files>` can be nested inside `<Directory>` sections to restrict the portion of the filesystem they apply to.

The *filename* argument should include a filename, or a wild-card string, where `?` matches any single character, and `*` matches any sequences of characters.

```
<Files "cat.html">
    # Insert stuff that applies to cat.html here
</Files>

<Files "?at.*">
    # This would apply to cat.html, bat.html, hat.php and so on.
</Files>
```

Regular expressions (↗ ../glossary.html#regex) can also be used, with the addition of the ~ character. For example:

```
<Files ~ "\.(gif|jpe?g|png)$">
    #...
</Files>
```

would match most common Internet graphics formats. `<FilesMatch>` is preferred, however.

Note that unlike `<Directory>` and `<Location>` sections, `<Files>` sections can be used inside `.htaccess` files. This allows users to control access to their own files, at a file-by-file level.

See also

- How <Directory>, <Location> and <Files> sections work for an explanation of how these different sections are combined when a request is received

<FilesMatch> Directive

Description:	Contains directives that apply to regular-expression matched filenames
Syntax:	`<FilesMatch regex> ... </FilesMatch>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `<FilesMatch>` directive limits the scope of the enclosed directives by filename, just as the `<Files>` directive does. However, it accepts a regular expression (↗ ../glossary.html#regex) . For example:

```
<FilesMatch ".+\.(gif|jpe?g|png)$">
    # ...
</FilesMatch>
```

would match most common Internet graphics formats.

> The `.+` at the start of the regex ensures that files named `.png`, or `.gif`, for example, are not matched.

From 2.4.8 onwards, named groups and backreferences are captured and written to the environment with the corresponding name prefixed with "MATCH_" and in upper case. This allows elements of files to be referenced from within expressions (↗ ../expr.html) and modules like `mod_rewrite`. In order to prevent confusion, numbered (unnamed) backreferences are ignored. Use named groups instead.

```
<FilesMatch "^(?<sitename>[^/]+)">
    Require ldap-group cn=%{env:MATCH_SITENAME},ou=combined,o=Example
</FilesMatch>
```

See also

- How <Directory>, <Location> and <Files> sections work for an explanation of how these different sections are combined when a request is received

FlushMaxPipelined Directive

Description:	Maximum number of pipelined responses above which they are flushed to the network
Syntax:	`FlushMaxPipelined number`
Default:	`FlushMaxPipelined 5`

Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	2.4.47 and later

This directive allows to configure the maximum number of pipelined responses, which remain pending so long as pipelined request are received. When the limit is reached, responses are forcibly flushed to the network in blocking mode, until passing under the limit again.

`FlushMaxPipelined` helps constraining memory usage. When set to `0` pipelining is disabled, when set to `-1` there is no limit (`FlushMaxThreshold` still applies).

FlushMaxThreshold Directive

Description:	Threshold above which pending data are flushed to the network
Syntax:	`FlushMaxThreshold` *`number-of-bytes`*
Default:	`FlushMaxThreshold 65536`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	2.4.47 and later

This directive allows to configure the threshold for pending output data (in bytes). When the limit is reached, data are forcibly flushed to the network in blocking mode, until passing under the limit again.

`FlushMaxThreshold` helps constraining memory usage. When set to `0` or a too small value there are actually no pending data, but for threaded MPMs there can be more threads busy waiting for the network thus less ones available to handle the other simultaneous connections.

ForceType Directive

Description:	Forces all matching files to be served with the specified media type in the HTTP Content-Type header field
Syntax:	`ForceType` *`media-type`*`|None`
Context:	directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

When placed into an `.htaccess` file or a `<Directory>`, or `<Location>` or `<Files>` section, this directive forces all matching files to be served with the content type identification given by *media-type*. For example, if you had a directory full of GIF files, but did not want to label them all with `.gif`, you might want to use:

```
ForceType image/gif
```

Note that this directive overrides other indirect media type associations defined in mime.types or via the `AddType`.

You can also override more general `ForceType` settings by using the value of `None`:

```
# force all files to be image/gif:
<Location "/images">
  ForceType image/gif
</Location>

# but normal mime-type associations here:
<Location "/images/mixed">
  ForceType None
</Location>
```

This directive primarily overrides the content types generated for static files served out of the filesystem. For resources other than static files, where the generator of the response typically specifies a Content-Type, this directive has no effect.

Note

When explicit directives such as `SetHandler` or `AddHandler` do not apply to the current request, the internal handler name normally set by those directives is set to match the content type specified by this directive. This is a historical behavior that some third-party modules (such as mod_php) may use "magic" content types used only to signal the module to take responsibility for the matching request. Configurations that rely on such "magic" types should be avoided by the use of `SetHandler` or `AddHandler`.

GprofDir Directive

Description:	Directory to write gmon.out profiling data to.
Syntax:	`GprofDir` *`/tmp/gprof/`*`|`*`/tmp/gprof/%`*
Context:	server config, virtual host
Status:	Core
Module:	core

When the server has been compiled with gprof profiling support, `GprofDir` causes `gmon.out` files to be written to the specified directory when the process exits. If the argument ends with a percent symbol ('%'), subdirectories are created for each process id.

This directive currently only works with the `prefork` MPM.

HostnameLookups Directive

Description:	Enables DNS lookups on client IP addresses
Syntax:	`HostnameLookups On|Off|Double`
Default:	`HostnameLookups Off`
Context:	server config, virtual host, directory
Status:	Core

Module:	core

This directive enables DNS lookups so that host names can be logged (and passed to CGIs/SSIs in `REMOTE_HOST`). The value `Double` refers to doing double-reverse DNS lookup. That is, after a reverse lookup is performed, a forward lookup is then performed on that result. At least one of the IP addresses in the forward lookup must match the original address. (In "tcpwrappers" terminology this is called `PARANOID`.)

Regardless of the setting, when `mod_authz_host` is used for controlling access by hostname, a double reverse lookup will be performed. This is necessary for security. Note that the result of this double-reverse isn't generally available unless you set `HostnameLookups Double`. For example, if only `HostnameLookups On` and a request is made to an object that is protected by hostname restrictions, regardless of whether the double-reverse fails or not, CGIs will still be passed the single-reverse result in `REMOTE_HOST`.

The default is `Off` in order to save the network traffic for those sites that don't truly need the reverse lookups done. It is also better for the end users because they don't have to suffer the extra latency that a lookup entails. Heavily loaded sites should leave this directive `Off`, since DNS lookups can take considerable amounts of time. The utility `logresolve`, compiled by default to the `bin` subdirectory of your installation directory, can be used to look up host names from logged IP addresses offline.

Finally, if you have hostname-based Require directives (↗ mod_authz_host.html#reqhost) , a hostname lookup will be performed regardless of the setting of `HostnameLookups`.

HttpProtocolOptions Directive

Description:	Modify restrictions on HTTP Request Messages
Syntax:	`HttpProtocolOptions [Strict|Unsafe] [RegisteredMethods|LenientMethods] [Allow0.9|Require1.0]`
Default:	`HttpProtocolOptions Strict LenientMethods Allow0.9`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	2.2.32 or 2.4.24 and later

This directive changes the rules applied to the HTTP Request Line (RFC 7230 §3.1.1 (↗ https://tools.ietf.org/html/rfc7230#section-3.1.1)) and the HTTP Request Header Fields (RFC 7230 §3.2 (↗ https://tools.ietf.org/html/rfc7230#section-3.2)), which are now applied by default or using the `Strict` option. Due to legacy modules, applications or custom user-agents which must be deprecated the `Unsafe` option has been added to revert to the legacy behaviors.

These rules are applied prior to request processing, so must be configured at the global or default (first) matching virtual host section, by IP/port interface (and not by name) to be honored.

The directive accepts three parameters from the following list of choices, applying the default to the ones not specified:

Strict|Unsafe

Prior to the introduction of this directive, the Apache HTTP Server request message parsers were tolerant of a number of forms of input which did not conform to the protocol. RFC 7230 §9.4 Request Splitting (↗ https://tools.ietf.org/html/rfc7230#section-9.4) and §9.5 Response Smuggling (↗ https://tools.ietf.org/html/rfc7230#section-9.5) call out only two of the potential risks of accepting non-conformant request messages, while RFC 7230 §3.5 (↗ https://tools.ietf.org/html/rfc7230#section-3.5) "Message Parsing Robustness" identify the risks of accepting obscure whitespace and request message formatting. As of the introduction of this directive, all grammar rules of the specification are enforced in the default `Strict` operating mode, and the strict whitespace suggested by section 3.5 is enforced and cannot be relaxed.

Security risks of Unsafe

Users are strongly cautioned against toggling the `Unsafe` mode of operation, particularly on outward-facing, publicly accessible server deployments. If an interface is required for faulty monitoring or other custom service consumers running on an intranet, users should toggle the Unsafe option only on a specific virtual host configured to service their internal private network.

Example of a request leading to HTTP 400 with Strict mode

```
# Missing CRLF
GET / HTTP/1.0\n\n
```

Command line tools and CRLF

Some tools need to be forced to use CRLF, otherwise httpd will return a HTTP 400 response like described in the above use case. For example, the **OpenSSL s_client needs the -crlf parameter to work properly**.
The `DumpIOInput` directive can help while reviewing the HTTP request to identify issues like the absence of CRLF.

RegisteredMethods|LenientMethods

RFC 7231 §4.1 (↗ https://tools.ietf.org/html/rfc7231#section-4.1) "Request Methods" "Overview" requires that origin servers shall respond with a HTTP 501 status code when an unsupported method is encountered in the request line. This already happens when the `LenientMethods` option is used, but administrators may wish to toggle the `RegisteredMethods` option and register any non-standard methods using the `RegisterHttpMethod` directive, particularly if the `Unsafe` option has been toggled.

Forward Proxy compatibility

The `RegisteredMethods` option should **not** be toggled for forward proxy hosts, as the methods supported by the origin servers are unknown to the proxy server.

Example of a request leading to HTTP 501 with LenientMethods mode

```
# Unknown HTTP method
WOW / HTTP/1.0\r\n\r\n

# Lowercase HTTP method
get / HTTP/1.0\r\n\r\n
```

Allow0.9|Require1.0

RFC 2616 §19.6 (↗ https://tools.ietf.org/html/rfc2616#section-19.6) "Compatibility With Previous Versions" had encouraged HTTP servers to support legacy HTTP/0.9 requests. RFC 7230 supersedes this with "The expectation to support HTTP/0.9 requests has been removed" and offers additional comments in RFC 7230 Appendix A (↗ https://tools.ietf.org/html/rfc7230#appendix-A) . The `Require1.0` option allows the user to remove support of the default `Allow0.9` option's behavior.

Example of a request leading to HTTP 400 with Require1.0 mode

```
# Unsupported HTTP version
GET /\r\n\r\n
```

Reviewing the messages logged to the `ErrorLog`, configured with `LogLevel` `debug` level, can help identify such faulty requests along with their origin. Users should pay particular attention to the 400 responses in the access log for invalid requests which were unexpectedly rejected.

<If> Directive

Description:	Contains directives that apply only if a condition is satisfied by a request at runtime
Syntax:	`<If expression> ... </If>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core
Compatibility:	Nested conditions are evaluated in 2.4.26 and later

The `<If>` directive evaluates an expression at runtime, and applies the enclosed directives if and only if the expression evaluates to true. For example:

```
<If "-z req('Host')">
```

would match HTTP/1.0 requests without a *Host:* header. Expressions may contain various shell-like operators for string comparison (`==`, `!=`, `<`, ...), integer comparison (`-eq`, `-ne`, ...), and others (`-n`, `-z`, `-f`, ...). It is also possible to use regular expressions,

```
<If "%{QUERY_STRING} =~ /(delete|commit)=.*?elem/">
```

shell-like pattern matches and many other operations. These operations can be done on request headers (`req`), environment variables (`env`), and a large number of other properties. The full documentation is available in Expressions in Apache HTTP Server (↗ ../expr.html) .

Only directives that support the directory context (↗ directive-dict.html#Context) can be used within this configuration section.

Certain variables, such as `CONTENT_TYPE` and other response headers, are set after <If> conditions have already been evaluated, and so will not be available to use in this directive.

Directives that take affect during configuration parsing, such as `Define`, `Include`, and `Error` cannot be made conditional by enclosing them in an if `<If>` configuration section. These sections are always part of the configuration, regardless of how they evaluate at runtime.

See also

- Expressions in Apache HTTP Server, for a complete reference and more examples.
- `<ElseIf>`
- `<Else>`
- How <Directory>, <Location>, <Files> sections work for an explanation of how these different sections are combined when a request is received. `<If>`, `<ElseIf>`, and `<Else>` are applied last.

<IfDefine> Directive

Description:	Encloses directives that will be processed only if a test is true at startup
Syntax:	`<IfDefine [!]parameter-name> ... </IfDefine>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `<IfDefine test>...</IfDefine>` section is used to mark directives that are conditional. The directives within an `<IfDefine>` section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

The *test* in the `<IfDefine>` section directive can be one of two forms:

- *parameter-name*
- `!`*parameter-name*

In the former case, the directives between the start and end markers are only processed if the parameter named *parameter-name* is defined. The second format reverses the test, and only processes the directives if *parameter-name* is **not** defined.

The *parameter-name* argument is a define as given on the `httpd` command line via `-Dparameter` at the time the server was started or by the `Define` directive.

`<IfDefine>` sections are nest-able, which can be used to implement simple multiple-parameter tests. Example:

```
httpd -DReverseProxy -DUseCache -DMemCache ...
```

```
<IfDefine ReverseProxy>
  LoadModule proxy_module   modules/mod_proxy.so
  LoadModule proxy_http_module   modules/mod_proxy_http.so
  <IfDefine UseCache>
    LoadModule cache_module   modules/mod_cache.so
    <IfDefine MemCache>
      LoadModule mem_cache_module   modules/mod_mem_cache.so
    </IfDefine>
    <IfDefine !MemCache>
      LoadModule cache_disk_module   modules/mod_cache_disk.so
    </IfDefine>
  </IfDefine>
</IfDefine>
```

<IfDirective> Directive

Description:	Encloses directives that are processed conditional on the presence or absence of a specific directive
Syntax:	`<IfDirective [!]directive-name> ... </IfDirective>`
Context:	server config, virtual host, directory, .htaccess

Override:	All
Status:	Core
Module:	core
Compatibility:	Available in 2.4.34 and later.

The `<IfDirective test>...</IfDirective>` section is used to mark directives that are conditional on the presence of a specific directive. The directives within an `<IfDirective>` section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

The *test* in the `<IfDirective>` section can be one of two forms:

- *directive-name*
- !*directive-name*

In the former case, the directives between the start and end markers are only processed if a directive of the given name is available at the time of processing. The second format reverses the test, and only processes the directives if *directive-name* is **not** available.

This section should only be used if you need to have one configuration file that works across multiple versions of `httpd`, regardless of whether a particular directive is available. In normal operation, directives need not be placed in `<IfDirective>` sections.

See also

- `<IfSection>`

<IfFile> Directive

Description:	Encloses directives that will be processed only if file exists at startup
Syntax:	`<IfFile [!]filename> ... </IfFile>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core
Compatibility:	Available in 2.4.34 and later.

The `<IfFile filename>...</IfFile>` section is used to mark directives that are conditional on the existence of a file on disk. The directives within an `<IfFile>` section are only processed if *filename* exists. If *filename* doesn't exist, everything between the start and end markers is ignored. *filename* can be an absolute path or a path relative to the server root.

The *filename* in the `<IfFile>` section directive can take the same forms as the *test* variable in the `<IfDefine>` section, i.e. the test can be negated if the `!` character is placed directly before *filename*.

If a relative *filename* is supplied, the check is `ServerRoot` relative. In the case where this directive occurs before the `ServerRoot`, the path will be checked relative to the compiled-in server root or the server root passed in on the command line via the `-d` parameter.

Warning

In 2.4.34, it is not possible to specify a *filename* with surrounding quotes. This would generate a parsing error at start-up. The main impact is that filenames with spaces can't be used. This behavior is fixed in 2.4.35.

<IfModule> Directive

Description:	Encloses directives that are processed conditional on the presence or absence of a specific module
Syntax:	`<IfModule [!]module-file|module-identifier> ... </IfModule>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core
Compatibility:	Module identifiers are available in version 2.1 and later.

The `<IfModule test>...</IfModule>` section is used to mark directives that are conditional on the presence of a specific module. The directives within an `<IfModule>` section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

The *test* in the `<IfModule>` section directive can be one of two forms:

- *module*
- !*module*

In the former case, the directives between the start and end markers are only processed if the module named *module* is included in Apache httpd -- either compiled in or dynamically loaded using `LoadModule`. The second format reverses the test, and only processes the directives if *module* is **not** included.

The *module* argument can be either the module identifier or the file name of the module, at the time it was compiled. For example, `rewrite_module` is the identifier and `mod_rewrite.c` is the file name. If a module consists of several source files, use the name of the file containing the string `STANDARD20_MODULE_STUFF`.

`<IfModule>` sections are nest-able, which can be used to implement simple multiple-module tests.

This section should only be used if you need to have one configuration file that works whether or not a specific module is available. In normal operation, directives need not be placed in `<IfModule>` sections.

<IfSection> Directive

Description:	Encloses directives that are processed conditional on the presence or absence of a specific section directive
Syntax:	`<IfSection [!]section-name> ... </IfSection>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Compatibility:	Available in 2.4.34 and later.

The `<IfSection test>...</IfSection>` section is used to mark directives that are conditional on the presence of a specific section directive. A section directive is any directive such as `<VirtualHost>` which encloses other directives, and has a directive name with a leading "<".

The directives within an `<IfSection>` section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

The *section-name* must be specified without either the leading "<" or closing ">". The *test* in the `<IfSection>` section can be one of two forms:

- *section-name*
- !*section-name*

In the former case, the directives between the start and end markers are only processed if a section directive of the given name is available at the time of processing. The second format reverses the test, and only processes the directives if *section-name* is **not** an available section directive.

For example:

```
<IfSection VirtualHost>
   ...
</IfSection>
```

This section should only be used if you need to have one configuration file that works across multiple versions of `httpd`, regardless of whether a particular section directive is available. In normal operation, directives need not be placed in `<IfSection>` sections.

See also

- `<IfDirective>`

Include Directive

Description:	Includes other configuration files from within the server configuration files
Syntax:	`Include` *`file-path|directory-path|wildcard`*
Context:	server config, virtual host, directory
Status:	Core
Module:	core
Compatibility:	Directory wildcard matching available in 2.3.6 and later

This directive allows inclusion of other configuration files from within the server configuration files.

Shell-style (`fnmatch()`) wildcard characters can be used in the filename or directory parts of the path to include several files at once, in alphabetical order. In addition, if `Include` points to a directory, rather than a file, Apache httpd will read all files in that directory and any subdirectory. However, including entire directories is not recommended, because it is easy to accidentally leave temporary files in a directory that can cause `httpd` to fail. Instead, we encourage you to use the wildcard syntax shown below, to include files that match a particular pattern, such as *.conf, for example.

The `Include` directive will **fail with an error** if a wildcard expression does not match any file. The `IncludeOptional` directive can be used if non-matching wildcards should be ignored.

The file path specified may be an absolute path, or may be relative to the `ServerRoot` directory.

Examples:

```
Include /usr/local/apache2/conf/ssl.conf
Include /usr/local/apache2/conf/vhosts/*.conf
```

Or, providing paths relative to your `ServerRoot` directory:

```
Include conf/ssl.conf
Include conf/vhosts/*.conf
```

Wildcards may be included in the directory or file portion of the path. This example will fail if there is no subdirectory in conf/vhosts that contains at least one *.conf file:

```
Include conf/vhosts/*/*.conf
```

Alternatively, the following command will just be ignored in case of missing files or directories:

```
IncludeOptional conf/vhosts/*/*.conf
```

See also

- `IncludeOptional`
- `apachectl`

IncludeOptional Directive

Description:	Includes other configuration files from within the server configuration files
Syntax:	`IncludeOptional` *`file-path|directory-path|wildcard`*
Context:	server config, virtual host, directory
Status:	Core
Module:	core
Compatibility:	Available in 2.3.6 and later. Not existent file paths without wildcards do not cause SyntaxError after 2.4.30

This directive allows inclusion of other configuration files from within the server configuration files. It works identically to the `Include` directive, but it will be silently ignored (instead of causing an error) if wildcards are used and they do not match any file or directory or if a file path does not exist on the file system.

See also

- `Include`
- `apachectl`

KeepAlive Directive

Description:	Enables HTTP persistent connections
Syntax:	`KeepAlive On|Off`
Default:	`KeepAlive On`
Context:	server config, virtual host
Status:	Core
Module:	core

The Keep-Alive extension to HTTP/1.0 and the persistent connection feature of HTTP/1.1 provide long-lived HTTP sessions which allow multiple requests to be sent over the same TCP connection. In some cases this has been shown to result in an almost 50% speedup in latency times for HTML documents with many images. To enable Keep-Alive connections, set `KeepAlive On`.

For HTTP/1.0 clients, Keep-Alive connections will only be used if they are specifically requested by a client. In addition, a Keep-Alive connection with an HTTP/1.0 client can only be used when the length of the content is known in advance. This implies that dynamic content such as CGI output, SSI pages, and server-generated directory listings will generally not use Keep-Alive connections to HTTP/1.0 clients. For HTTP/1.1 clients, persistent connections are the default unless otherwise specified. If the client requests it, chunked encoding will be used in order to send content of unknown length over persistent connections.

When a client uses a Keep-Alive connection, it will be counted as a single "request" for the `MaxConnectionsPerChild` directive, regardless of how many requests are sent using the connection.

See also

- `MaxKeepAliveRequests`

KeepAliveTimeout Directive

Description:	Amount of time the server will wait for subsequent requests on a persistent connection
Syntax:	`KeepAliveTimeout` *`num`*`[ms]`
Default:	`KeepAliveTimeout 5`
Context:	server config, virtual host
Status:	Core
Module:	core

The number of seconds Apache httpd will wait for a subsequent request before closing the connection. By adding a postfix of ms the timeout can be also set in milliseconds. Once a request has been received, the timeout value specified by the `Timeout` directive applies.

Setting `KeepAliveTimeout` to a high value may cause performance problems in heavily loaded servers. The higher the timeout, the more server processes will be kept occupied waiting on connections with idle clients.

If `KeepAliveTimeout` is **not** set for a name-based virtual host, the value of the first defined virtual host best matching the local IP and port will be used.

<Limit> Directive

Description:	Restrict enclosed access controls to only certain HTTP methods
Syntax:	`<Limit` *`method`* `[`*`method`*`] ... > ... </Limit>`
Context:	directory, .htaccess
Override:	AuthConfig, Limit
Status:	Core
Module:	core

Access controls are normally effective for **all** access methods, and this is the usual desired behavior. **In the general case, access control directives should not be placed within a `<Limit>` section.**

The purpose of the `<Limit>` directive is to restrict the effect of the access controls to the nominated HTTP methods. For all other methods, the access restrictions that are enclosed in the `<Limit>` bracket **will have no effect**. The following example applies the access control only to the methods `POST`, `PUT`, and `DELETE`, leaving all other methods unprotected:

```
<Limit POST PUT DELETE>
  Require valid-user
</Limit>
```

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`. **The method name is case-sensitive.** If `GET` is used, it will also restrict `HEAD` requests. The `TRACE` method cannot be limited (see `TraceEnable`).

A `<LimitExcept>` section should always be used in preference to a `<Limit>` section when restricting access, since a `<LimitExcept>` section provides protection against arbitrary methods.

The `<Limit>` and `<LimitExcept>` directives may be nested. In this case, each successive level of `<Limit>` or `<LimitExcept>` directives must further restrict the set of methods to which access controls apply.

When using `<Limit>` or `<LimitExcept>` directives with the `Require` directive, note that the first `Require` to succeed authorizes the request, regardless of the presence of other `Require` directives.

For example, given the following configuration, all users will be authorized for `POST` requests, and the `Require group editors` directive will be ignored in all cases:

```
<LimitExcept GET>
  Require valid-user
</LimitExcept>
<Limit POST>
```

```
  Require group editors
</Limit>
```

<LimitExcept> Directive

Description:	Restrict access controls to all HTTP methods except the named ones
Syntax:	`<LimitExcept method [method] ... > ... </LimitExcept>`
Context:	directory, .htaccess
Override:	AuthConfig, Limit
Status:	Core
Module:	core

`<LimitExcept>` and `</LimitExcept>` are used to enclose a group of access control directives which will then apply to any HTTP access method **not** listed in the arguments; i.e., it is the opposite of a `<Limit>` section and can be used to control both standard and nonstandard/unrecognized methods. See the documentation for `<Limit>` for more details.

For example:

```
<LimitExcept POST GET>
  Require valid-user
</LimitExcept>
```

LimitInternalRecursion Directive

Description:	Determine maximum number of internal redirects and nested subrequests
Syntax:	`LimitInternalRecursion number [number]`
Default:	`LimitInternalRecursion 10`
Context:	server config, virtual host
Status:	Core
Module:	core

An internal redirect happens, for example, when using the `Action` directive, which internally redirects the original request to a CGI script. A subrequest is Apache httpd's mechanism to find out what would happen for some URI if it were requested. For example, `mod_dir` uses subrequests to look for the files listed in the `DirectoryIndex` directive.

`LimitInternalRecursion` prevents the server from crashing when entering an infinite loop of internal redirects or subrequests. Such loops are usually caused by misconfigurations.

The directive stores two different limits, which are evaluated on per-request basis. The first *number* is the maximum number of internal redirects that may follow each other. The second *number* determines how deeply subrequests may be nested. If you specify only one *number*, it will be assigned to both limits.

```
LimitInternalRecursion 5
```

LimitRequestBody Directive

Description:	Restricts the total size of the HTTP request body sent from the client
Syntax:	`LimitRequestBody bytes`
Default:	`LimitRequestBody 1073741824`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core
Compatibility:	In Apache HTTP Server 2.4.53 and earlier, the default value was 0 (unlimited)

This directive specifies the number of *bytes* that are allowed in a request body. A value of *0* means unlimited.

The `LimitRequestBody` directive allows the user to set a limit on the allowed size of an HTTP request message body within the context in which the directive is given (server, per-directory, per-file or per-location). If the client request exceeds that limit, the server will return an error response instead of servicing the request. The size of a normal request message body will vary greatly depending on the nature of the resource and the methods allowed on that resource. CGI scripts typically use the message body for retrieving form information. Implementations of the `PUT` method will require a value at least as large as any representation that the server wishes to accept for that resource.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks.

If, for example, you are permitting file upload to a particular location and wish to limit the size of the uploaded file to 100K, you might use the following directive:

```
LimitRequestBody 102400
```

LimitRequestFields Directive

Description:	Limits the number of HTTP request header fields that will be accepted from the client
Syntax:	`LimitRequestFields number`
Default:	`LimitRequestFields 100`
Context:	server config, virtual host
Status:	Core
Module:	core

Setting *number* at 0 means unlimited. The default value is defined by the compile-time constant `DEFAULT_LIMIT_REQUEST_FIELDS` (100 as distributed).

The `LimitRequestFields` directive allows the server administrator to modify the limit on the number of request header fields allowed in an HTTP request. A server needs this value to be larger than the number of fields that a normal client request might include. The number of request header fields used by a client rarely exceeds 20, but this may vary among different client implementations, often depending upon the extent to which a user has configured their browser to support detailed content negotiation. Optional HTTP extensions are often expressed using request header fields.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks. The value should be increased if normal clients see an error response from the server that indicates too many fields were sent in the request.

For example:

```
LimitRequestFields 50
```

> **Warning**
>
> When name-based virtual hosting is used, the value for this directive is taken from the default (first-listed) virtual host for the local IP and port combination.

LimitRequestFieldSize Directive

Description:	Limits the size of the HTTP request header allowed from the client
Syntax:	`LimitRequestFieldSize` *`bytes`*
Default:	`LimitRequestFieldSize 8190`
Context:	server config, virtual host
Status:	Core
Module:	core

This directive specifies the number of *bytes* that will be allowed in an HTTP request header.

The `LimitRequestFieldSize` directive allows the server administrator to set the limit on the allowed size of an HTTP request header field. A server needs this value to be large enough to hold any one header field from a normal client request. The size of a normal request header field will vary greatly among different client implementations, often depending upon the extent to which a user has configured their browser to support detailed content negotiation. SPNEGO authentication headers can be up to 12392 bytes.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks.

For example:

```
LimitRequestFieldSize 4094
```

> Under normal conditions, the value should not be changed from the default.

> **Warning**
>
> When name-based virtual hosting is used, the value for this directive is taken from the default (first-listed) virtual host best matching the current IP address and port combination.

LimitRequestLine Directive

Description:	Limit the size of the HTTP request line that will be accepted from the client
Syntax:	`LimitRequestLine` *`bytes`*
Default:	`LimitRequestLine 8190`
Context:	server config, virtual host
Status:	Core
Module:	core

This directive sets the number of *bytes* that will be allowed on the HTTP request-line.

The `LimitRequestLine` directive allows the server administrator to set the limit on the allowed size of a client's HTTP request-line. Since the request-line consists of the HTTP method, URI, and protocol version, the `LimitRequestLine` directive places a restriction on the length of a request-URI allowed for a request on the server. A server needs this value to be large enough to hold any of its resource names, including any information that might be passed in the query part of a `GET` request.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks.

For example:

```
LimitRequestLine 4094
```

> Under normal conditions, the value should not be changed from the default.

> **Warning**
>
> When name-based virtual hosting is used, the value for this directive is taken from the default (first-listed) virtual host best matching the current IP address and port combination.

LimitXMLRequestBody Directive

Description:	Limits the size of an XML-based request body
Syntax:	`LimitXMLRequestBody` *`bytes`*
Default:	`LimitXMLRequestBody 1000000`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Limit (in bytes) on the maximum size of an XML-based request body. A value of `0` will apply a hard limit (depending on 32bit vs 64bit system) allowing for XML escaping within the bounds of the system addressable memory, but it exists for compatibility only and is not recommended since it does not account for memory consumed elsewhere or concurrent requests, which might result in an overall system out-of-memory.

Example:

```
# Limit of 1 MiB
LimitXMLRequestBody 1073741824
```

<Location> Directive

Description:	Applies the enclosed directives only to matching URLs
Syntax:	`<Location URL-path|URL> ... </Location>`
Context:	server config, virtual host
Status:	Core
Module:	core

The `<Location>` directive limits the scope of the enclosed directives by URL. It is similar to the `<Directory>` directive, and starts a subsection which is terminated with a `</Location>` directive. `<Location>` sections are processed in the order they appear in the configuration file, after the `<Directory>` sections and `.htaccess` files are read, and after the `<Files>` sections.

`<Location>` sections operate completely outside the filesystem. This has several consequences. Most importantly, `<Location>` directives should not be used to control access to filesystem locations. Since several different URLs may map to the same filesystem location, such access controls may by circumvented.

The enclosed directives will be applied to the request if the path component of the URL meets *any* of the following criteria:

- The specified location matches exactly the path component of the URL.
- The specified location, which ends in a forward slash, is a prefix of the path component of the URL (treated as a context root).
- The specified location, with the addition of a trailing slash, is a prefix of the path component of the URL (also treated as a context root).

In the example below, where no trailing slash is used, requests to /private1, /private1/ and /private1/file.txt will have the enclosed directives applied, but /private1other would not.

```
<Location "/private1">
    #  ...
</Location>
```

In the example below, where a trailing slash is used, requests to /private2/ and /private2/file.txt will have the enclosed directives applied, but /private2 and /private2other would not.

```
<Location "/private2/">
    # ...
</Location>
```

When to use `<Location>`

Use `<Location>` to apply directives to content that lives outside the filesystem. For content that lives in the filesystem, use `<Directory>` and `<Files>`. An exception is `<Location "/">`, which is an easy way to apply a configuration to the entire server.

For all origin (non-proxy) requests, the URL to be matched is a URL-path of the form `/path/`. *No scheme, hostname, port, or query string may be included.* For proxy requests, the URL to be matched is of the form `scheme://servername/path`, and you must include the prefix.

The URL may use wildcards. In a wild-card string, `?` matches any single character, and `*` matches any sequences of characters. Neither wildcard character matches a / in the URL-path.

Regular expressions (↗ ../glossary.html#regex) can also be used, with the addition of the ~ character. For example:

```
<Location ~ "/(extra|special)/data">
    #...
</Location>
```

would match URLs that contained the substring `/extra/data` or `/special/data`. The directive `<LocationMatch>` behaves identical to the regex version of `<Location>`, and is preferred, for the simple reason that ~ is hard to distinguish from `-` in many fonts.

The `<Location>` functionality is especially useful when combined with the `SetHandler` directive. For example, to enable status requests but allow them only from browsers at `example.com`, you might use:

```
<Location "/status">
  SetHandler server-status
  Require host example.com
</Location>
```

Note about / (slash)

The slash character has special meaning depending on where in a URL it appears. People may be used to its behavior in the filesystem where multiple adjacent slashes are frequently collapsed to a single slash (*i.e.*, `/home///foo` is the same as `/home/foo`). In URL-space this is not necessarily true if directive `MergeSlashes` has been set to "OFF". The `<LocationMatch>` directive and the regex version of `<Location>` require you to explicitly specify multiple slashes if the slashes are not being merged. For example, `<LocationMatch "^/abc">` would match the request URL `/abc` but not the request URL `//abc`. The (non-regex) `<Location>` directive behaves similarly when used for proxy requests. But when (non-regex) `<Location>` is used for non-proxy requests it will implicitly match multiple slashes with a single slash. For example, if you specify `<Location "/abc/def">` and the request is to `/abc//def` then it will match.

See also

- How <Directory>, <Location> and <Files> sections work for an explanation of how these different sections are combined when a request is received.
- `LocationMatch`

<LocationMatch> Directive

Description:	Applies the enclosed directives only to regular-expression matching URLs
Syntax:	`<LocationMatch regex> ... </LocationMatch>`
Context:	server config, virtual host
Status:	Core
Module:	core

The <LocationMatch> directive limits the scope of the enclosed directives by URL, in an identical manner to <Location>. However, it takes a regular expression (↗ ../glossary.html#regex) as an argument instead of a simple string. For example:

```
<LocationMatch "/(extra|special)/data">
    # ...
</LocationMatch>
```

would match URLs that contained the substring `/extra/data` or `/special/data`.

> If the intent is that a URL **starts with** `/extra/data`, rather than merely **contains** `/extra/data`, prefix the regular expression with a `^` to require this.
>
> ```
> <LocationMatch "^/(extra|special)/data">
> ```

From 2.4.8 onwards, named groups and backreferences are captured and written to the environment with the corresponding name prefixed with "MATCH_" and in upper case. This allows elements of URLs to be referenced from within expressions (↗ ../expr.html) and modules like `mod_rewrite`. In order to prevent confusion, numbered (unnamed) backreferences are ignored. Use named groups instead.

```
<LocationMatch "^/combined/(?<sitename>[^/]+)">
    Require ldap-group cn=%{env:MATCH_SITENAME},ou=combined,o=Example
</LocationMatch>
```

> **Note about / (slash)**
>
> The slash character has special meaning depending on where in a URL it appears. People may be used to its behavior in the filesystem where multiple adjacent slashes are frequently collapsed to a single slash (*i.e.*, `/home///foo` is the same as `/home/foo`). In URL-space this is not necessarily true if directive `MergeSlashes` has been set to "OFF". The `<LocationMatch>` directive and the regex version of `<Location>` require you to explicitly specify multiple slashes if the slashes are not being merged. For example, `<LocationMatch "^/abc">` would match the request URL `/abc` but not the request URL `//abc`. The (non-regex) `<Location>` directive behaves similarly when used for proxy requests. But when (non-regex) `<Location>` is used for non-proxy requests it will implicitly match multiple slashes with a single slash. For example, if you specify `<Location "/abc/def">` and the request is to `/abc//def` then it will match.

See also

- How <Directory>, <Location> and <Files> sections work for an explanation of how these different sections are combined when a request is received

LogLevel Directive

Description:	Controls the verbosity of the ErrorLog
Syntax:	`LogLevel [module:]level [module:level] ...`
Default:	`LogLevel warn`
Context:	server config, virtual host, directory
Status:	Core
Module:	core
Compatibility:	Per-module and per-directory configuration is available in Apache HTTP Server 2.3.6 and later

`LogLevel` adjusts the verbosity of the messages recorded in the error logs (see `ErrorLog` directive). The following *level*s are available, in order of decreasing significance:

Level	Description	Example
`emerg`	Emergencies - system is unusable.	"Child cannot open lock file. Exiting"
`alert`	Action must be taken immediately.	"getpwuid: couldn't determine user name from uid"
`crit`	Critical Conditions.	"socket: Failed to get a socket, exiting child"
`error`	Error conditions.	"Premature end of script headers"
`warn`	Warning conditions.	"child process 1234 did not exit, sending another SIGHUP"
`notice`	Normal but significant condition.	"httpd: caught SIGBUS, attempting to dump core in ..."
`info`	Informational.	"Server seems busy, (you may need to increase StartServers, or Min/MaxSpareServers)..."
`debug`	Debug-level messages	"Opening config file ..."
`trace1`	Trace messages	"proxy: FTP: control connection complete"
`trace2`	Trace messages	"proxy: CONNECT: sending the CONNECT request to the remote proxy"
`trace3`	Trace messages	"openssl: Handshake: start"
`trace4`	Trace messages	"read from buffered SSL brigade, mode 0, 17 bytes"
`trace5`	Trace messages	"map lookup FAILED: map=rewritemap key=keyname"
`trace6`	Trace messages	"cache lookup FAILED, forcing new map lookup"
`trace7`	Trace messages, dumping large amounts of data	"| 0000: 02 23 44 30 13 40 ac 34 df 3d bf 9a 19 49 39 15 |"
`trace8`	Trace messages, dumping large amounts of data	"| 0000: 02 23 44 30 13 40 ac 34 df 3d bf 9a 19 49 39 15 |"

When a particular level is specified, messages from all other levels of higher significance will be reported as well. *E.g.*, when `LogLevel info` is specified, then messages with log levels of `notice` and `warn` will also be posted.

Using a level of at least `crit` is recommended.

For example:

```
LogLevel notice
```

> **Note**
>
> When logging to a regular file, messages of the level `notice` cannot be suppressed and thus are always logged. However, this doesn't apply when logging is done using `syslog`.

Specifying a level without a module name will reset the level for all modules to that level. Specifying a level with a module name will set the level for that module only. It is possible to use the module source file name, the module identifier, or the module identifier with the trailing `_module` omitted as module specification. This means the following three specifications are equivalent:

```
LogLevel info ssl:warn
LogLevel info mod_ssl.c:warn
LogLevel info ssl_module:warn
```

It is also possible to change the level per directory:

```
LogLevel info
<Directory "/usr/local/apache/htdocs/app">
  LogLevel debug
</Directory>
```

Per directory loglevel configuration only affects messages that are logged after the request has been parsed and that are associated with the request. Log messages which are associated with the connection or the server are not affected.

See also

- `ErrorLog`
- `ErrorLogFormat`
- Apache HTTP Server Log Files

MaxKeepAliveRequests Directive

Description:	Number of requests allowed on a persistent connection
Syntax:	`MaxKeepAliveRequests` *`number`*
Default:	`MaxKeepAliveRequests 100`
Context:	server config, virtual host
Status:	Core
Module:	core

The `MaxKeepAliveRequests` directive limits the number of requests allowed per connection when `KeepAlive` is on. If it is set to `0`, unlimited requests will be allowed. We recommend that this setting be kept to a high value for maximum server performance.

For example:

```
MaxKeepAliveRequests 500
```

MaxRangeOverlaps Directive

Description:	Number of overlapping ranges (eg: `100-200,150-300`) allowed before returning the complete resource
Syntax:	`MaxRangeOverlaps default | unlimited | none |` *`number-of-ranges`*
Default:	`MaxRangeOverlaps 20`
Context:	server config, virtual host, directory
Status:	Core
Module:	core
Compatibility:	Available in Apache HTTP Server 2.3.15 and later

The `MaxRangeOverlaps` directive limits the number of overlapping HTTP ranges the server is willing to return to the client. If more overlapping ranges than permitted are requested, the complete resource is returned instead.

default
Limits the number of overlapping ranges to a compile-time default of 20.

none
No overlapping Range headers are allowed.

unlimited
The server does not limit the number of overlapping ranges it is willing to satisfy.

number-of-ranges
A positive number representing the maximum number of overlapping ranges the server is willing to satisfy.

MaxRangeReversals Directive

Description:	Number of range reversals (eg: `100-200,50-70`) allowed before returning the complete resource
Syntax:	`MaxRangeReversals default | unlimited | none |` *`number-of-ranges`*
Default:	`MaxRangeReversals 20`
Context:	server config, virtual host, directory
Status:	Core
Module:	core
Compatibility:	Available in Apache HTTP Server 2.3.15 and later

The `MaxRangeReversals` directive limits the number of HTTP Range reversals the server is willing to return to the client. If more ranges reversals than permitted are requested, the complete resource is returned instead.

default
Limits the number of range reversals to a compile-time default of 20.

none
No Range reversals headers are allowed.

unlimited
The server does not limit the number of range reversals it is willing to satisfy.

number-of-ranges
A positive number representing the maximum number of range reversals the server is willing to satisfy.

MaxRanges Directive

Description:	Number of ranges allowed before returning the complete resource
Syntax:	`MaxRanges default | unlimited | none | number-of-ranges`
Default:	`MaxRanges 200`
Context:	server config, virtual host, directory
Status:	Core
Module:	core
Compatibility:	Available in Apache HTTP Server 2.3.15 and later

The `MaxRanges` directive limits the number of HTTP ranges the server is willing to return to the client. If more ranges than permitted are requested, the complete resource is returned instead.

default
  Limits the number of ranges to a compile-time default of 200.

none
  Range headers are ignored.

unlimited
  The server does not limit the number of ranges it is willing to satisfy.

number-of-ranges
  A positive number representing the maximum number of ranges the server is willing to satisfy.

MergeSlashes Directive

Description:	Controls whether the server merges consecutive slashes in URLs.
Syntax:	`MergeSlashes ON|OFF`
Default:	`MergeSlashes ON`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	Added in 2.4.39

By default, the server merges (or collapses) multiple consecutive slash ('/') characters in the path component of the request URL.

When mapping URL's to the filesystem, these multiple slashes are not significant. However, URL's handled other ways, such as by CGI or proxy, might prefer to retain the significance of multiple consecutive slashes. In these cases `MergeSlashes` can be set to *OFF* to retain the multiple consecutive slashes, which is the legacy behavior.

When set to "OFF", regular expressions used in the configuration file that match the path component of the URL (`LocationMatch`, `RewriteRule`, ...) need to take into account multiple consecutive slashes. Non regular expression based `Location` always operate against a URL with merged slashes and cannot differentiate between multiple slashes.

MergeTrailers Directive

Description:	Determines whether trailers are merged into headers
Syntax:	`MergeTrailers [on|off]`
Default:	`MergeTrailers off`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	2.4.11 and later

This directive controls whether HTTP trailers are copied into the internal representation of HTTP headers. This merging occurs when the request body has been completely consumed, long after most header processing would have a chance to examine or modify request headers.

This option is provided for compatibility with releases prior to 2.4.11, where trailers were always merged.

Mutex Directive

Description:	Configures mutex mechanism and lock file directory for all or specified mutexes
Syntax:	`Mutex mechanism [default|mutex-name] ... [OmitPID]`
Default:	`Mutex default`
Context:	server config
Status:	Core
Module:	core
Compatibility:	Available in Apache HTTP Server 2.3.4 and later

The `Mutex` directive sets the mechanism, and optionally the lock file location, that httpd and modules use to serialize access to resources. Specify `default` as the second argument to change the settings for all mutexes; specify a mutex name (see table below) as the second argument to override defaults only for that mutex.

The `Mutex` directive is typically used in the following exceptional situations:

- change the mutex mechanism when the default mechanism selected by APR has a functional or performance problem
- change the directory used by file-based mutexes when the default directory does not support locking

Supported modules

This directive only configures mutexes which have been registered with the core server using the `ap_mutex_register()` API. All modules bundled with httpd support the `Mutex` directive, but third-party modules may not. Consult the documentation of the third-party module, which must indicate the mutex name(s) which can be configured if this directive is supported.

The following mutex *mechanisms* are available:

- `default | yes`
  This selects the default locking implementation, as determined by APR (↗ ../glossary.html#apr) . The default locking implementation can be displayed by running `httpd` with the `-V` option.

- `none | no`
  This effectively disables the mutex, and is only allowed for a mutex if the module indicates that it is a valid choice. Consult the module documentation for more information.

- `posixsem`
  This is a mutex variant based on a Posix semaphore.

  > **Warning**
  > The semaphore ownership is not recovered if a thread in the process holding the mutex segfaults, resulting in a hang of the web server.

- `sysvsem`
  This is a mutex variant based on a SystemV IPC semaphore.

  > **Warning**
  > It is possible to "leak" SysV semaphores if processes crash before the semaphore is removed.

  > **Security**
  > The semaphore API allows for a denial of service attack by any CGIs running under the same uid as the webserver (*i.e.*, all CGIs, unless you use something like `suexec` or `cgiwrapper`).

- `sem`
  This selects the "best" available semaphore implementation, choosing between Posix and SystemV IPC semaphores, in that order.

- `pthread`
  This is a mutex variant based on cross-process Posix thread mutexes.

  > **Warning**
  > On most systems, if a child process terminates abnormally while holding a mutex that uses this implementation, the server will deadlock and stop responding to requests. When this occurs, the server will require a manual restart to recover.
  > Solaris and Linux are notable exceptions as they provide a mechanism which usually allows the mutex to be recovered after a child process terminates abnormally while holding a mutex.
  > If your system is POSIX compliant or if it implements the `pthread_mutexattr_setrobust_np()` function, you may be able to use the `pthread` option safely.

- `fcntl:/path/to/mutex`
  This is a mutex variant where a physical (lock-)file and the `fcntl()` function are used as the mutex.

  > **Warning**
  > When multiple mutexes based on this mechanism are used within multi-threaded, multi-process environments, deadlock errors (EDEADLK) can be reported for valid mutex operations if `fcntl()` is not thread-aware, such as on Solaris.

- `flock:/path/to/mutex`
  This is similar to the `fcntl:/path/to/mutex` method with the exception that the `flock()` function is used to provide file locking.

- `file:/path/to/mutex`
  This selects the "best" available file locking implementation, choosing between `fcntl` and `flock`, in that order.

Most mechanisms are only available on selected platforms, where the underlying platform and APR (↗ ../glossary.html#apr) support it. Mechanisms which aren't available on all platforms are *posixsem*, *sysvsem*, *sem*, *pthread*, *fcntl*, *flock*, and *file*.

With the file-based mechanisms *fcntl* and *flock*, the path, if provided, is a directory where the lock file will be created. The default directory is httpd's run-time file directory relative to `ServerRoot`. Always use a local disk filesystem for `/path/to/mutex` and never a directory residing on a NFS- or AFS-filesystem. The basename of the file will be the mutex type, an optional instance string provided by the module, and unless the `OmitPID` keyword is specified, the process id of the httpd parent process will be appended to make the file name unique, avoiding conflicts when multiple httpd instances share a lock file directory. For example, if the mutex name is `mpm-accept` and the lock file directory is `/var/httpd/locks`, the lock file name for the httpd instance with parent process id 12345 would be `/var/httpd/locks/mpm-accept.12345`.

> **Security**
> It is best to *avoid* putting mutex files in a world-writable directory such as `/var/tmp` because someone could create a denial of service attack and prevent the server from starting by creating a lockfile with the same name as the one the server will try to create.

The following table documents the names of mutexes used by httpd and bundled modules.

Mutex name	Module(s)	Protected resource
`mpm-accept`	`prefork` and `worker` MPMs	incoming connections, to avoid the thundering herd problem; for more information, refer to the performance tuning documentation
`authdigest-client`	`mod_auth_digest`	client list in shared memory
`authdigest-opaque`	`mod_auth_digest`	counter in shared memory
`ldap-cache`	`mod_ldap`	LDAP result cache
`rewrite-map`	`mod_rewrite`	communication with external mapping programs, to avoid intermixed I/O from multiple requests
`ssl-cache`	`mod_ssl`	SSL session cache
`ssl-stapling`	`mod_ssl`	OCSP stapling response cache
`watchdog-callback`	`mod_watchdog`	callback function of a particular client module

The `OmitPID` keyword suppresses the addition of the httpd parent process id from the lock file name.

In the following example, the mutex mechanism for the MPM accept mutex will be changed from the compiled-in default to `fcntl`, with the associated lock file created in directory `/var/httpd/locks`. The mutex mechanism for all other mutexes will be changed from the compiled-in default to `sysvsem`.

```
Mutex sysvsem default
Mutex fcntl:/var/httpd/locks mpm-accept
```

NameVirtualHost Directive

Description:	DEPRECATED: Designates an IP address for name-virtual hosting
Syntax:	`NameVirtualHost` *addr*`[:`*port*`]`
Context:	server config
Status:	Core
Module:	core

Prior to 2.3.11, `NameVirtualHost` was required to instruct the server that a particular IP address and port combination was usable as a name-based virtual host. In 2.3.11 and later, any time an IP address and port combination is used in multiple virtual hosts, name-based virtual hosting is automatically enabled for that address.

This directive currently has no effect.

See also

- Virtual Hosts documentation

Options Directive

Description:	Configures what features are available in a particular directory
Syntax:	`Options [+|-]`*option* `[[+|-]`*option*`] ...`
Default:	`Options FollowSymlinks`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Core
Module:	core
Compatibility:	The default was changed from All to FollowSymlinks in 2.3.11

The `Options` directive controls which server features are available in a particular directory.

option can be set to `None`, in which case none of the extra features are enabled, or one or more of the following:

`All`
All options except for `MultiViews`.

`ExecCGI`
Execution of CGI scripts using `mod_cgi` is permitted.

`FollowSymLinks`
The server will follow symbolic links in this directory. This is the default setting.

> Even though the server follows the symlink it does *not* change the pathname used to match against `<Directory>` sections.
> The `FollowSymLinks` and `SymLinksIfOwnerMatch` `Options` work only in `<Directory>` sections or `.htaccess` files.
> Omitting this option should not be considered a security restriction, since symlink testing is subject to race conditions that make it circumventable.

`Includes`
Server-side includes provided by `mod_include` are permitted.

`IncludesNOEXEC`
Server-side includes are permitted, but the `#exec cmd` and `#exec cgi` are disabled. It is still possible to `#include virtual` CGI scripts from `ScriptAlias`ed directories.

`Indexes`
If a URL which maps to a directory is requested and there is no `DirectoryIndex` (*e.g.*, `index.html`) in that directory, then `mod_autoindex` will return a formatted listing of the directory.

`MultiViews`
Content negotiated "MultiViews" are allowed using `mod_negotiation`.

> **Note**
> This option gets ignored if set anywhere other than `<Directory>`, as `mod_negotiation` needs real resources to compare against and evaluate from.

`SymLinksIfOwnerMatch`
The server will only follow symbolic links for which the target file or directory is owned by the same user id as the link.

> **Note**
> The `FollowSymLinks` and `SymLinksIfOwnerMatch` `Options` work only in `<Directory>` sections or `.htaccess` files.
> This option should not be considered a security restriction, since symlink testing is subject to race conditions that make it circumventable.

Normally, if multiple `Options` could apply to a directory, then the most specific one is used and others are ignored; the options are not merged. (See how sections are merged (↗ ../sections.html#merging) .) However if *all* the options on the `Options` directive are preceded by a `+` or `-` symbol, the options are merged. Any options preceded by a `+` are added to the options currently in force, and any options preceded by a `-` are removed from the options currently in force.

> **Note**
> Mixing `Options` with a `+` or `-` with those without is not valid syntax and will be rejected during server startup by the syntax check with an abort.

For example, without any `+` and `-` symbols:

```
<Directory "/web/docs">
  Options Indexes FollowSymLinks
</Directory>

<Directory "/web/docs/spec">
  Options Includes
</Directory>
```

then only `Includes` will be set for the `/web/docs/spec` directory. However if the second `Options` directive uses the `+` and `-` symbols:

```
<Directory "/web/docs">
  Options Indexes FollowSymLinks
</Directory>

<Directory "/web/docs/spec">
  Options +Includes -Indexes
</Directory>
```

then the options `FollowSymLinks` and `Includes` are set for the `/web/docs/spec` directory.

> **Note**
>
> Using `-IncludesNOEXEC` or `-Includes` disables server-side includes completely regardless of the previous setting.

The default in the absence of any other settings is `FollowSymlinks`.

Protocol Directive

Description:	Protocol for a listening socket
Syntax:	`Protocol` *`protocol`*
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	Available in Apache 2.1.5 and later. On Windows, from Apache 2.3.3 and later.

This directive specifies the protocol used for a specific listening socket. The protocol is used to determine which module should handle a request and to apply protocol specific optimizations with the `AcceptFilter` directive.

This directive not required for most configurations. If not specified, `https` is the default for port 443 and `http` the default for all other ports. The protocol is used to determine which module should handle a request, and to apply protocol specific optimizations with the `AcceptFilter` directive.

For example, if you are running `https` on a non-standard port, specify the protocol explicitly:

```
Protocol https
```

You can also specify the protocol using the `Listen` directive.

See also

- `AcceptFilter`
- `Listen`

Protocols Directive

Description:	Protocols available for a server/virtual host
Syntax:	`Protocols` *`protocol`* `...`
Default:	`Protocols http/1.1`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	Only available from Apache 2.4.17 and later.

This directive specifies the list of protocols supported for a server/virtual host. The list determines the allowed protocols a client may negotiate for this server/host.

You need to set protocols if you want to extend the available protocols for a server/host. By default, only the http/1.1 protocol (which includes the compatibility with 1.0 and 0.9 clients) is allowed.

For example, if you want to support HTTP/2 for a server with TLS, specify:

```
Protocols h2 http/1.1
```

Valid protocols are `http/1.1` for http and https connections, `h2` on https connections and `h2c` for http connections. Modules may enable more protocols.

It is safe to specify protocols that are unavailable/disabled. Such protocol names will simply be ignored.

Protocols specified in base servers are inherited for virtual hosts only if the virtual host has no own Protocols directive. Or, the other way around, Protocols directives in virtual hosts replace any such directive in the base server.

See also

- `ProtocolsHonorOrder`

ProtocolsHonorOrder Directive

Description:	Determines if order of Protocols determines precedence during negotiation
Syntax:	`ProtocolsHonorOrder On|Off`
Default:	`ProtocolsHonorOrder On`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	Only available from Apache 2.4.17 and later.

This directive specifies if the server should honor the order in which the `Protocols` directive lists protocols.

If configured Off, the client supplied list order of protocols has precedence over the order in the server configuration.

With `ProtocolsHonorOrder` set to `on` (default), the client ordering does not matter and only the ordering in the server settings influences the outcome of the protocol negotiation.

See also

- `Protocols`

QualifyRedirectURL Directive

Description:	Controls whether the REDIRECT_URL environment variable is fully qualified
Syntax:	`QualifyRedirectURL On|Off`
Default:	`QualifyRedirectURL Off`
Context:	server config, virtual host, directory
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Directive supported in 2.4.18 and later. 2.4.17 acted as if 'QualifyRedirectURL On' was configured.

This directive controls whether the server will ensure that the REDIRECT_URL environment variable is fully qualified. By default, the variable contains the verbatim URL requested by the client, such as "/index.html". With `QualifyRedirectURL On`, the same request would result in a value such as "http://www.example.com/index.html".

Even without this directive set, when a request is issued against a fully qualified URL, REDIRECT_URL will remain fully qualified.

ReadBufferSize Directive

Description:	Size of the buffers used to read data
Syntax:	`ReadBufferSize` *`bytes`*
Default:	`ReadBufferSize 8192`
Context:	server config, virtual host, directory
Status:	Core
Module:	core
Compatibility:	2.4.27 and later

This directive allows to configure the size (in bytes) of the memory buffer used to read data from the network or files.

A larger buffer can increase peformances with larger data, but consumes more memory per connection. The minimum configurable size is *1024*.

RegexDefaultOptions Directive

Description:	Allow to configure global/default options for regexes
Syntax:	`RegexDefaultOptions [none] [+|-]`*`option`*` [[+|-]`*`option`*`] ...`
Default:	`RegexDefaultOptions DOTALL DOLLAR_ENDONLY`
Context:	server config
Status:	Core
Module:	core
Compatibility:	Only available from Apache 2.4.30 and later.

This directive adds some default behavior to ANY regular expression used afterwards.

Any option preceded by a '+' is added to the already set options.
Any option preceded by a '-' is removed from the already set options.
Any option without a '+' or a '-' will be set, removing any other already set option.
The `none` keyword resets any already set options.

option can be:

`ICASE`
: Use a case-insensitive match.

`EXTENDED`
: Perl's /x flag, ignore (unescaped-)spaces and comments in the pattern.

`DOTALL`
: Perl's /s flag, '.' matches newline characters.

`DOLLAR_ENDONLY`
: '$' matches at end of subject string only.

```
# Add the ICASE option for all regexes by default
RegexDefaultOptions +ICASE
...
# Remove the default DOLLAR_ENDONLY option, but keep any other one
RegexDefaultOptions -DOLLAR_ENDONLY
...
# Set the DOTALL option only, resetting any other one
RegexDefaultOptions DOTALL
...
# Reset all defined options
RegexDefaultOptions none
...
```

RegisterHttpMethod Directive

Description:	Register non-standard HTTP methods
Syntax:	`RegisterHttpMethod` *`method`*` [`*`method`*` [...]]`
Context:	server config
Status:	Core

Module:	core
Compatibility:	Available in Apache HTTP Server 2.4.24 and later

This directive may be used to register additional HTTP methods. This is necessary if non-standard methods need to be used with directives that accept method names as parameters, or to allow particular non-standard methods to be used via proxy or CGI script when the server has been configured to only pass recognized methods to modules.

See also

- `HTTPProtocolOptions`
- `AllowMethods`

RLimitCPU Directive

Description:	Limits the CPU consumption of processes launched by Apache httpd children
Syntax:	`RLimitCPU` *`seconds`*`|max [`*`seconds`*`|max]`
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes and the second parameter sets the maximum resource limit. Either parameter can be a number, or `max` to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as `root` or in the initial startup phase.

This applies to processes forked from Apache httpd children servicing requests, not the Apache httpd children themselves. This includes CGI scripts and SSI exec commands, but not any processes forked from the Apache httpd parent, such as piped logs.

CPU resource limits are expressed in seconds per process.

See also

- `RLimitMEM`
- `RLimitNPROC`

RLimitMEM Directive

Description:	Limits the memory consumption of processes launched by Apache httpd children
Syntax:	`RLimitMEM` *`bytes`*`|max [`*`bytes`*`|max]`
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes and the second parameter sets the maximum resource limit. Either parameter can be a number, or `max` to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as `root` or in the initial startup phase.

This applies to processes forked from Apache httpd children servicing requests, not the Apache httpd children themselves. This includes CGI scripts and SSI exec commands, but not any processes forked from the Apache httpd parent, such as piped logs.

Memory resource limits are expressed in bytes per process.

See also

- `RLimitCPU`
- `RLimitNPROC`

RLimitNPROC Directive

Description:	Limits the number of processes that can be launched by processes launched by Apache httpd children
Syntax:	`RLimitNPROC` *`number`*`|max [`*`number`*`|max]`
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes, and the second parameter sets the maximum resource limit. Either parameter can be a number, or `max` to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as `root` or in the initial startup phase.

This applies to processes forked from Apache httpd children servicing requests, not the Apache httpd children themselves. This includes CGI scripts and SSI exec commands, but not any processes forked from the Apache httpd parent, such as piped logs.

Process limits control the number of processes per user.

Note

If CGI processes are **not** running under user ids other than the web server user id, this directive will limit the number of processes that the server itself can create. Evidence of this situation will be indicated by **`cannot fork`** messages in the `error_log`.

See also

- `RLimitMEM`
- `RLimitCPU`

ScriptInterpreterSource Directive

Description:	Technique for locating the interpreter for CGI scripts
Syntax:	`ScriptInterpreterSource Registry|Registry-Strict|Script`
Default:	`ScriptInterpreterSource Script`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Win32 only.

This directive is used to control how Apache httpd finds the interpreter used to run CGI scripts. The default setting is `Script`. This causes Apache httpd to use the interpreter pointed to by the shebang line (first line, starting with `#!`) in the script. On Win32 systems this line usually looks like:

```
#!C:/Perl/bin/perl.exe
```

or, if `perl` is in the `PATH`, simply:

```
#!perl
```

Setting `ScriptInterpreterSource Registry` will cause the Windows Registry tree `HKEY_CLASSES_ROOT` to be searched using the script file extension (e.g., `.pl`) as a search key. The command defined by the registry subkey `Shell\ExecCGI\Command` or, if it does not exist, by the subkey `Shell\Open\Command` is used to open the script file. If the registry keys cannot be found, Apache httpd falls back to the behavior of the `Script` option.

Security

Be careful when using `ScriptInterpreterSource Registry` with `ScriptAlias`'ed directories, because Apache httpd will try to execute **every** file within this directory. The `Registry` setting may cause undesired program calls on files which are typically not executed. For example, the default open command on `.htm` files on most Windows systems will execute Microsoft Internet Explorer, so any HTTP request for an `.htm` file existing within the script directory would start the browser in the background on the server. This is a good way to crash your system within a minute or so.

The option `Registry-Strict` does the same thing as `Registry` but uses only the subkey `Shell\ExecCGI\Command`. The `ExecCGI` key is not a common one. It must be configured manually in the windows registry and hence prevents accidental program calls on your system.

SeeRequestTail Directive

Description:	Determine if mod_status displays the first 63 characters of a request or the last 63, assuming the request itself is greater than 63 chars.
Syntax:	`SeeRequestTail On|Off`
Default:	`SeeRequestTail Off`
Context:	server config
Status:	Core
Module:	core
Compatibility:	Available in Apache httpd 2.2.7 and later.

`mod_status` with `ExtendedStatus On` displays the actual request being handled. For historical purposes, only 63 characters of the request are actually stored for display purposes. This directive controls whether the first 63 characters are stored (the previous behavior and the default) or if the last 63 characters are. This is only applicable, of course, if the length of the request is 64 characters or greater.

If Apache httpd is handling `GET /disk1/storage/apache/htdocs/images/imagestore1/food/apples.jpg HTTP/1.1` `mod_status` displays as follows:

Off (default)	GET /disk1/storage/apache/htdocs/images/imagestore1/food/apples
On	orage/apache/htdocs/images/imagestore1/food/apples.jpg HTTP/1.1

ServerAdmin Directive

Description:	Email address that the server includes in error messages sent to the client
Syntax:	`ServerAdmin` *`email-address`*`|`*`URL`*
Context:	server config, virtual host
Status:	Core
Module:	core

The `ServerAdmin` sets the contact address that the server includes in any error messages it returns to the client. If the `httpd` doesn't recognize the supplied argument as an URL, it assumes, that it's an *email-address* and prepends it with `mailto:` in hyperlink targets. However, it's recommended to actually use an email address, since there are a lot of CGI scripts that make that assumption. If you want to use an URL, it should point to another server under your control. Otherwise users may not be able to contact you in case of errors.

It may be worth setting up a dedicated address for this, e.g.

```
ServerAdmin www-admin@foo.example.com
```

as users do not always mention that they are talking about the server!

ServerAlias Directive

Description:	Alternate names for a host used when matching requests to name-virtual hosts
Syntax:	`ServerAlias` *`hostname`* `[`*`hostname`*`] ...`
Context:	virtual host
Status:	Core
Module:	core

The `ServerAlias` directive sets the alternate names for a host, for use with name-based virtual hosts (↗ ../vhosts/name-based.html) . The `ServerAlias` may include wildcards, if appropriate.

```
<VirtualHost *:80>
  ServerName server.example.com
  ServerAlias server server2.example.com server2
  ServerAlias *.example.com
  UseCanonicalName Off
  # ...
</VirtualHost>
```

Name-based virtual hosts for the best-matching set of `<virtualhost>`s are processed in the order they appear in the configuration. The first matching `ServerName` or `ServerAlias` is used, with no different precedence for wildcards (nor for ServerName vs. ServerAlias).

The complete list of names in the `<VirtualHost>` directive are treated just like a (non wildcard) `ServerAlias`.

See also

- `UseCanonicalName`
- Apache HTTP Server Virtual Host documentation

ServerName Directive

Description:	Hostname and port that the server uses to identify itself
Syntax:	`ServerName [`*`scheme`*`://]`*`domain-name`*`|`*`ip-address`*`[:`*`port`*`]`
Context:	server config, virtual host
Status:	Core
Module:	core

The `ServerName` directive sets the request scheme, hostname and port that the server uses to identify itself.

`ServerName` is used (possibly in conjunction with `ServerAlias`) to uniquely identify a virtual host, when using name-based virtual hosts (↗ ../vhosts/name-based.html) .

Additionally, this is used when creating self-referential redirection URLs when `UseCanonicalName` is set to a non-default value.

For example, if the name of the machine hosting the web server is `simple.example.com`, but the machine also has the DNS alias `www.example.com` and you wish the web server to be so identified, the following directive should be used:

```
ServerName www.example.com
```

The `ServerName` directive may appear anywhere within the definition of a server. However, each appearance overrides the previous appearance (within that server).

If no `ServerName` is specified, the server attempts to deduce the client visible hostname by first asking the operating system for the system hostname, and if that fails, performing a reverse lookup on an IP address present on the system.

If no port is specified in the `ServerName`, then the server will use the port from the incoming request. For optimal reliability and predictability, you should specify an explicit hostname and port using the `ServerName` directive.

If you are using name-based virtual hosts (↗ ../vhosts/name-based.html) , the `ServerName` inside a `<VirtualHost>` section specifies what hostname must appear in the request's `Host:` header to match this virtual host.

Sometimes, the server runs behind a device that processes SSL, such as a reverse proxy, load balancer or SSL offload appliance. When this is the case, specify the `https://` scheme and the port number to which the clients connect in the `ServerName` directive to make sure that the server generates the correct self-referential URLs.

See the description of the `UseCanonicalName` and `UseCanonicalPhysicalPort` directives for settings which determine whether self-referential URLs (e.g., by the `mod_dir` module) will refer to the specified port, or to the port number given in the client's request.

Failure to set `ServerName` to a name that your server can resolve to an IP address will result in a startup warning. `httpd` will then use whatever hostname it can determine, using the system's `hostname` command. This will almost never be the hostname you actually want.

```
httpd: Could not reliably determine the server's fully qualified domain name, using rocinante.local for ServerName
```

See also

- Issues Regarding DNS and Apache HTTP Server
- Apache HTTP Server virtual host documentation
- `UseCanonicalName`
- `UseCanonicalPhysicalPort`
- `ServerAlias`

ServerPath Directive

Description:	Legacy URL pathname for a name-based virtual host that is accessed by an incompatible browser
Syntax:	`ServerPath `*`URL-path`*
Context:	virtual host
Status:	Core
Module:	core

The `ServerPath` directive sets the legacy URL pathname for a host, for use with name-based virtual hosts (↗ ../vhosts/) .

See also

- Apache HTTP Server Virtual Host documentation

ServerRoot Directive

Description:	Base directory for the server installation
Syntax:	`ServerRoot `*`directory-path`*

Default:	`ServerRoot /usr/local/apache`
Context:	server config
Status:	Core
Module:	core

The `ServerRoot` directive sets the directory in which the server lives. Typically it will contain the subdirectories `conf/` and `logs/`. Relative paths in other configuration directives (such as `Include` or `LoadModule`, for example) are taken as relative to this directory.

```
ServerRoot "/home/httpd"
```

The default location of `ServerRoot` may be modified by using the `--prefix` argument to `configure` (↗ ../programs/configure.html) , and most third-party distributions of the server have a different default location from the one listed above.

See also

- the `-d` option to `httpd`
- the security tips for information on how to properly set permissions on the `ServerRoot`

ServerSignature Directive

Description:	Configures the footer on server-generated documents
Syntax:	`ServerSignature On|Off|EMail`
Default:	`ServerSignature Off`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `ServerSignature` directive allows the configuration of a trailing footer line under server-generated documents (error messages, `mod_proxy` ftp directory listings, `mod_info` output, ...). The reason why you would want to enable such a footer line is that in a chain of proxies, the user often has no possibility to tell which of the chained servers actually produced a returned error message.

The `Off` setting, which is the default, suppresses the footer line. The `On` setting simply adds a line with the server version number and `ServerName` of the serving virtual host, and the `EMail` setting additionally creates a "mailto:" reference to the `ServerAdmin` of the referenced document.

The details of the server version number presented are controlled by the `ServerTokens` directive.

See also

- `ServerTokens`

ServerTokens Directive

Description:	Configures the `Server` HTTP response header
Syntax:	`ServerTokens Major|Minor|Min[imal]|Prod[uctOnly]|OS|Full`
Default:	`ServerTokens Full`
Context:	server config
Status:	Core
Module:	core

This directive controls whether `Server` response header field which is sent back to clients includes a description of the generic OS-type of the server as well as information about compiled-in modules.

`ServerTokens Full` (or not specified)
: Server sends (*e.g.*): `Server: Apache/2.4.2 (Unix) PHP/4.2.2 MyMod/1.2`

`ServerTokens Prod[uctOnly]`
: Server sends (*e.g.*): `Server: Apache`

`ServerTokens Major`
: Server sends (*e.g.*): `Server: Apache/2`

`ServerTokens Minor`
: Server sends (*e.g.*): `Server: Apache/2.4`

`ServerTokens Min[imal]`
: Server sends (*e.g.*): `Server: Apache/2.4.2`

`ServerTokens OS`
: Server sends (*e.g.*): `Server: Apache/2.4.2 (Unix)`

This setting applies to the entire server, and cannot be enabled or disabled on a virtualhost-by-virtualhost basis.

This directive also controls the information presented by the `ServerSignature` directive.

> Setting `ServerTokens` to less than `minimal` is not recommended because it makes it more difficult to debug interoperational problems. Also note that disabling the Server: header does nothing at all to make your server more secure. The idea of "security through obscurity" is a myth and leads to a false sense of safety.

See also

- `ServerSignature`

SetHandler Directive

Description:	Forces all matching files to be processed by a handler
Syntax:	`SetHandler` *`handler-name`*`|none|`*`expression`*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo

Status:	Core
Module:	core
Compatibility:	expression argument 2.4.19 and later

When placed into an `.htaccess` file or a `<Directory>` or `<Location>` section, this directive forces all matching files to be parsed through the handler (↗ ../handler.html) given by *handler-name*. For example, if you had a directory you wanted to be parsed entirely as imagemap rule files, regardless of extension, you might put the following into an `.htaccess` file in that directory:

```
SetHandler imap-file
```

Another example: if you wanted to have the server display a status report whenever a URL of `http://servername/status` was called, you might put the following into `httpd.conf`:

```
<Location "/status">
  SetHandler server-status
</Location>
```

You could also use this directive to configure a particular handler for files with a particular file extension. For example:

```
<FilesMatch "\.php$">
    SetHandler application/x-httpd-php
</FilesMatch>
```

String-valued expressions can be used to reference per-request variables, including backreferences to named regular expressions:

```
<LocationMatch ^/app/(?<sub>[^/]+)/>
     SetHandler "proxy:unix:/var/run/app_%{env:MATCH_sub}.sock|fcgi://localhost:8080"
</LocationMatch>
```

You can override an earlier defined `SetHandler` directive by using the value `None`.

Note

Because `SetHandler` overrides default handlers, normal behavior such as handling of URLs ending in a slash (/) as directories or index files is suppressed.

See also

- `AddHandler`

SetInputFilter Directive

Description:	Sets the filters that will process client requests and POST input
Syntax:	`SetInputFilter` *`filter`*`[;`*`filter`*`...]`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

The `SetInputFilter` directive sets the filter or filters which will process client requests and POST input when they are received by the server. This is in addition to any filters defined elsewhere, including the `AddInputFilter` directive.

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content.

See also

- Filters documentation

SetOutputFilter Directive

Description:	Sets the filters that will process responses from the server
Syntax:	`SetOutputFilter` *`filter`*`[;`*`filter`*`...]`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

The `SetOutputFilter` directive sets the filters which will process responses from the server before they are sent to the client. This is in addition to any filters defined elsewhere, including the `AddOutputFilter` directive.

For example, the following configuration will process all files in the `/www/data/` directory for server-side includes.

```
<Directory "/www/data/">
  SetOutputFilter INCLUDES
</Directory>
```

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content.

See also

- Filters documentation

StrictHostCheck Directive

Description:	Controls whether the server requires the requested hostname be listed enumerated in the virtual host handling the request

Syntax:	`StrictHostCheck ON|OFF`
Default:	`StrictHostCheck OFF`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	Added in 2.4.49

By default, the server will respond to requests for any hostname, including requests addressed to unexpected or unconfigured hostnames. While this is convenient, it is sometimes desirable to limit what hostnames a backend application handles since it will often generate self-referential responses.

By setting `StrictHostCheck` to *ON*, the server will return an HTTP 400 error if the requested hostname hasn't been explicitly listed by either `ServerName` or `ServerAlias` in the virtual host that best matches the details of the incoming connection.

This directive also allows matching of the requested hostname to hostnames specified within the opening `VirtualHost` tag, which is a relatively obscure configuration mechanism that acts like additional `ServerAlias` entries.

This directive has no affect in non-default virtual hosts. The value inherited from the global server configuration, or the default virtualhost for the ip:port the underlying connection, determine the effective value.

TimeOut Directive

Description:	Amount of time the server will wait for certain events before failing a request
Syntax:	`TimeOut` *`seconds`*
Default:	`TimeOut 60`
Context:	server config, virtual host
Status:	Core
Module:	core

The `TimeOut` directive defines the length of time Apache httpd will wait for I/O in various circumstances:

- When reading data from the client, the length of time to wait for a TCP packet to arrive if the read buffer is empty.

  For initial data on a new connection, this directive doesn't take effect until after any configured `AcceptFilter` has passed the new connection to the server.

- When writing data to the client, the length of time to wait for an acknowledgement of a packet if the send buffer is full.
- In `mod_cgi` and `mod_cgid`, the length of time to wait for any individual block of output from a CGI script.
- In `mod_ext_filter`, the length of time to wait for output from a filtering process.
- In `mod_proxy`, the default timeout value if `ProxyTimeout` is not configured.

TraceEnable Directive

Description:	Determines the behavior on `TRACE` requests
Syntax:	`TraceEnable` *`[on|off|extended]`*
Default:	`TraceEnable on`
Context:	server config, virtual host
Status:	Core
Module:	core

This directive overrides the behavior of `TRACE` for both the core server and `mod_proxy`. The default `TraceEnable on` permits `TRACE` requests per RFC 2616, which disallows any request body to accompany the request. `TraceEnable off` causes the core server and `mod_proxy` to return a `405` (Method not allowed) error to the client.

Finally, for testing and diagnostic purposes only, request bodies may be allowed using the non-compliant `TraceEnable extended` directive. The core (as an origin server) will restrict the request body to 64Kb (plus 8Kb for chunk headers if `Transfer-Encoding: chunked` is used). The core will reflect the full headers and all chunk headers with the response body. As a proxy server, the request body is not restricted to 64Kb.

Note

Despite claims to the contrary, enabling the `TRACE` method does not expose any security vulnerability in Apache httpd. The `TRACE` method is defined by the HTTP/1.1 specification and implementations are expected to support it.

UnDefine Directive

Description:	Undefine the existence of a variable
Syntax:	`UnDefine` *`parameter-name`*
Context:	server config
Status:	Core
Module:	core

Undoes the effect of a `Define` or of passing a `-D` argument to `httpd`.

This directive can be used to toggle the use of `<IfDefine>` sections without needing to alter `-D` arguments in any startup scripts.

Variable names may not contain colon ":" characters, to avoid clashes with `RewriteMap`'s syntax.

Virtual Host scope and pitfalls

While this directive is supported in virtual host context, the changes it makes are visible to any later configuration directives, beyond any enclosing virtual host.

See also

- `Define`
- `IfDefine`

UseCanonicalName Directive

Description:	Configures how the server determines its own name and port

Syntax:	`UseCanonicalName On|Off|DNS`
Default:	`UseCanonicalName Off`
Context:	server config, virtual host, directory
Status:	Core
Module:	core

In many situations Apache httpd must construct a *self-referential* URL -- that is, a URL that refers back to the same server. With `UseCanonicalName On` Apache httpd will use the hostname and port specified in the `ServerName` directive to construct the canonical name for the server. This name is used in all self-referential URLs, and for the values of `SERVER_NAME` and `SERVER_PORT` in CGIs.

With `UseCanonicalName Off` Apache httpd will form self-referential URLs using the hostname and port supplied by the client if any are supplied (otherwise it will use the canonical name, as defined above). These values are the same that are used to implement name-based virtual hosts (↗ ../vhosts/name-based.html) and are available with the same clients. The CGI variables `SERVER_NAME` and `SERVER_PORT` will be constructed from the client supplied values as well.

An example where this may be useful is on an intranet server where you have users connecting to the machine using short names such as `www`. You'll notice that if the users type a shortname and a URL which is a directory, such as `http://www/splat`, *without the trailing slash*, then Apache httpd will redirect them to `http://www.example.com/splat/`. If you have authentication enabled, this will cause the user to have to authenticate twice (once for `www` and once again for `www.example.com` -- see the FAQ on this subject for more information (↗ http://wiki.apache.org/httpd/FAQ#Why_does_Apache_ask_for_my_password_twice_before_serving_a_file.3F)). But if `UseCanonicalName` is set `Off`, then Apache httpd will redirect to `http://www/splat/`.

There is a third option, `UseCanonicalName DNS`, which is intended for use with mass IP-based virtual hosting to support ancient clients that do not provide a `Host:` header. With this option, Apache httpd does a reverse DNS lookup on the server IP address that the client connected to in order to work out self-referential URLs.

Warning

If CGIs make assumptions about the values of `SERVER_NAME`, they may be broken by this option. The client is essentially free to give whatever value they want as a hostname. But if the CGI is only using `SERVER_NAME` to construct self-referential URLs, then it should be just fine.

See also

- `UseCanonicalPhysicalPort`
- `ServerName`
- `Listen`

UseCanonicalPhysicalPort Directive

Description:	Configures how the server determines its own port
Syntax:	`UseCanonicalPhysicalPort On|Off`
Default:	`UseCanonicalPhysicalPort Off`
Context:	server config, virtual host, directory
Status:	Core
Module:	core

In many situations Apache httpd must construct a *self-referential* URL -- that is, a URL that refers back to the same server. With `UseCanonicalPhysicalPort On`, Apache httpd will, when constructing the canonical port for the server to honor the `UseCanonicalName` directive, provide the actual physical port number being used by this request as a potential port. With `UseCanonicalPhysicalPort Off`, Apache httpd will not ever use the actual physical port number, instead relying on all configured information to construct a valid port number.

Note

The ordering of the lookup when the physical port is used is as follows:

`UseCanonicalName On`

1. Port provided in `Servername`
2. Physical port
3. Default port

`UseCanonicalName Off | DNS`

1. Parsed port from `Host:` header
2. Physical port
3. Port provided in `Servername`
4. Default port

With `UseCanonicalPhysicalPort Off`, the physical ports are removed from the ordering.

See also

- `UseCanonicalName`
- `ServerName`
- `Listen`

<VirtualHost> Directive

Description:	Contains directives that apply only to a specific hostname or IP address
Syntax:	`<VirtualHost addr[:port] [addr[:port]] ...> ... </VirtualHost>`
Context:	server config
Status:	Core
Module:	core

`<VirtualHost>` and `</VirtualHost>` are used to enclose a group of directives that will apply only to a particular virtual host. Any directive that is allowed in a virtual host context may be used. When the server receives a request for a document on a particular virtual host, it uses the configuration directives enclosed in the `<VirtualHost>` section. *Addr* can be any of the following, optionally followed by a colon and a port number (or *):

- The IP address of the virtual host;
- A fully qualified domain name for the IP address of the virtual host (not recommended);
- The character `*`, which acts as a wildcard and matches any IP address.
- The string `_default_`, which is an alias for `*`

```
<VirtualHost 10.1.2.3:80>
  ServerAdmin webmaster@host.example.com
  DocumentRoot "/www/docs/host.example.com"
  ServerName host.example.com
  ErrorLog "logs/host.example.com-error_log"
  TransferLog "logs/host.example.com-access_log"
</VirtualHost>
```

IPv6 addresses must be specified in square brackets because the optional port number could not be determined otherwise. An IPv6 example is shown below:

```
<VirtualHost [2001:db8::a00:20ff:fea7:ccea]:80>
  ServerAdmin webmaster@host.example.com
  DocumentRoot "/www/docs/host.example.com"
  ServerName host.example.com
  ErrorLog "logs/host.example.com-error_log"
  TransferLog "logs/host.example.com-access_log"
</VirtualHost>
```

Each Virtual Host must correspond to a different IP address, different port number, or a different host name for the server, in the former case the server machine must be configured to accept IP packets for multiple addresses. (If the machine does not have multiple network interfaces, then this can be accomplished with the `ifconfig alias` command -- if your OS supports it).

Note

The use of `<VirtualHost>` does **not** affect what addresses Apache httpd listens on. You may need to ensure that Apache httpd is listening on the correct addresses using `Listen`.

A `ServerName` should be specified inside each `<VirtualHost>` block. If it is absent, the `ServerName` from the "main" server configuration will be inherited.

When a request is received, the server first maps it to the best matching `<VirtualHost>` based on the local IP address and port combination only. Non-wildcards have a higher precedence. If no match based on IP and port occurs at all, the "main" server configuration is used.

If multiple virtual hosts contain the best matching IP address and port, the server selects from these virtual hosts the best match based on the requested hostname. If no matching name-based virtual host is found, then the first listed virtual host that matched the IP address will be used. As a consequence, the first listed virtual host for a given IP address and port combination is the default virtual host for that IP and port combination.

Security

See the security tips (↗ ../misc/security_tips.html) document for details on why your security could be compromised if the directory where log files are stored is writable by anyone other than the user that starts the server.

See also

- Apache HTTP Server Virtual Host documentation
- Issues Regarding DNS and Apache HTTP Server
- Setting which addresses and ports Apache HTTP Server uses
- How <Directory>, <Location> and <Files> sections work for an explanation of how these different sections are combined when a request is received

Comments

Notice:
This is not a Q&A section. Comments placed here should be pointed towards suggestions on improving the documentation or server, and may be removed by our moderators if they are either implemented or considered invalid/off-topic. Questions on how to manage the Apache HTTP Server should be directed at either our IRC channel, #httpd, on Libera.chat, or sent to our mailing lists.