HttpCoreModule

WARNING: this article is obsoleted. Please refer to http://nginx.org/en/docs/ for the latest official documentation.

Contents

Synopsis

Controls core features of Nginx's HTTP processing.

Directives

aio

Syntax:	**aio** `on` | `off` | `sendfile`
Default:	*off*
Context:	http server location
Appeared in:	0.8.11
Reference:	aio (http://nginx.org/en/docs/http/ngx_http_core_module.html#aio)

This directive is usable as of Linux kernel 2.6.22. For Linux it is required to use directio, this automatically disables sendfile support.

```
location /video {
    aio on;
    directio 512;
    output_buffers 1 128k;
}
```

In FreeBSD before 5.2.1 and Nginx 0.8.12 you must disable sendfile support.

```
location /video {
    aio on;
    sendfile off;
    output_buffers 1 128k;
}
```

As of FreeBSD 5.2.1 and Nginx 0.8.12 you can use it along with sendfile.

```
location /video {
    aio sendfile;
    sendfile on;
    tcp_nopush on;
}
```

alias

Syntax:	**alias** *path*
Default:	
Context:	location
Reference:	alias (http://nginx.org/en/docs/http/ngx_http_core_module.html#alias)

This directive assigns a path to be used as the basis for serving requests for the indicated location. Note that it may look similar to the `root` directive at first sight, but the document root doesn't change, just the file system path used for the request. The location part of the request is **dropped** in the request Nginx issues. Let's see this in action. Consider the following example.

```
location  /i/ {
  alias  /spool/w3/images/;
}
```

A request for "/i/top.gif" will instruct Nginx to serve the file "/spool/w3/images/top.gif". As you can see, **only** the part of the URI **after** the location is appended. The location itself, in this case "/i/", is dropped. With a `root` directive the **full** path is appended, i.e., in the above example it would have been, "/spool/w3/images/i/top.gif" — hence including **also** the location "/i/".

Aliases can also be used in a location specified by a regex.

For example:

```
location ~ ^/download/(.*)$ {
  alias /home/website/files/$1;
}
```

The request "/download/book.pdf" will return the file "/home/website/files/book.pdf". Note again that only part of the request URI **after** the location is appended to the path defined by `alias`.

It is possible to use variables in the replacement path.

Note that there is a longstanding bug (http://trac.nginx.org/nginx/ticket/97) that `alias` and `try_files` don't work together.

chunked_transfer_encoding

Syntax:	**chunked_transfer_encoding** `on` | `off`
Default:	*on*
Context:	http server location
Reference:	chunked_transfer_encoding (http://nginx.org/en/docs/http/ngx_http_core_module.html#chunked_transfer_encoding)

This directive (0.7.66+) sets whether chunked encoding is enabled in responses (only valid for connections using HTTP 1.1 or later).

client_body_in_file_only

Syntax:	**client_body_in_file_only** `on` | `clean` | `off`
Default:	*off*
Context:	http server location
Reference:	client_body_in_file_only (http://nginx.org/en/docs/http/ngx_http_core_module.html#client_body_in_file_only)

The directive forces nginx to always store a client request body into a temporary disk file even if the body is actually of 0 size.

Please note that the file will **NOT** be removed at request completion if the directive is enabled.

This directive can be used for debugging and for the `$r->request_body_file` method in the Embedded Perl module.

client_body_in_single_buffer

Syntax:	**client_body_in_single_buffer** `on|off`
Default:	*off*
Context:	http server location
Reference:	client_body_in_single_buffer (http://nginx.org/en/docs/http/ngx_http_core_module.html#client_body_in_single_buffer)

The directive(0.7.58+) specifies whether to keep the whole body in a single client request buffer. The directive is recommended when using the variable $request_body to reduce the operations of copying.

Note that when the request body cannot be hold in a single buffer (see client_body_buffer_size), the body will still touch the disk.

client_body_buffer_size

Syntax:	**client_body_buffer_size** *size*
Default:	*8k/16k*
Context:	http server location
Reference:	client_body_buffer_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#client_body_buffer_size)

The directive specifies the client request body buffer size.

If the request body size is more than the buffer size, then the entire (or partial) request body is written into a temporary file.

The default size is equal to page size times 2. Depending on the platform, the page size is either 8K or 16K.

When the *Content-Length* request header specifies a smaller size value than the buffer size, then Nginx will use the smaller one. As a result, Nginx will **not** always allocate a buffer of this buffer size for every request.

client_body_temp_path

Syntax:	**client_body_temp_path** *path* [*level1* [*level2* [*level3*]]]
Default:	*client_body_temp*
Context:	http server location
Reference:	client_body_temp_path (http://nginx.org/en/docs/http/ngx_http_core_module.html#client_body_temp_path)

The directive assigns the directory for storing the temporary files in it with the body of the request.

In the `dir-path` a hierarchy of subdirectories up to three levels are possible.

For example

```
client_body_temp_path  /spool/nginx/client_temp 1 2;
```

The directory structure will be like this:

```
/spool/nginx/client_temp/7/45/00000123457
```

client_body_timeout

Syntax:	**client_body_timeout** *time*
Default:	*60s*
Context:	http server location
Reference:	client_body_timeout (http://nginx.org/en/docs/http/ngx_http_core_module.html#client_body_timeout)

Directive sets the read timeout for the request body from client.

The timeout is set only if a body is not get in one readstep. If after this time the client send nothing, nginx returns error "Request time out" (408).

client_header_buffer_size

Syntax:	**client_header_buffer_size** *size*
Default:	*1k*
Context:	http server
Reference:	client_header_buffer_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#client_header_buffer_size)

Directive sets the headerbuffer size for the request header from client.

For the overwhelming majority of requests it is completely sufficient with a buffer size of 1K.

However if a big cookie is in the request-header or the request has come from a wap-client the header can not be placed in 1K, therefore, the request-header or a line of request-header is not located completely in this buffer nginx allocate a bigger buffer, the size of the bigger buffer can be set with the instruction large_client_header_buffers.

client_header_timeout

Syntax:	**client_header_timeout** *time*
Default:	*60s*
Context:	http server
Reference:	client_header_timeout (http://nginx.org/en/docs/http/ngx_http_core_module.html#client_header_timeout)

Specifies how long to wait for the client to send a request header (e.g.: `GET / HTTP/1.1`).

This timeout is reached only if a header is not received in one read *(needs clarification)*. If the client has not sent anything within this timeout period, nginx returns the HTTP status code 408 ("Request timed out")

client_max_body_size

Syntax:	**client_max_body_size** *size*
Default:	*1m*
Context:	http server location
Reference:	client_max_body_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#client_max_body_size)

Specifies the maximum accepted body size of a client request, as indicated by the request header `Content-Length`.

If the stated content length is greater than this size, then the client receives the HTTP error code 413 ("Request Entity Too Large"). It should be noted that web browsers do not usually know how to properly display such an HTTP error.

Set to 0 to disable.

connection_pool_size

Syntax:	**connection_pool_size** *size*
Default:	*256*
Context:	http server
Reference:	connection_pool_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#connection_pool_size)

The directive is used to allocate memory per connection. The pool is used for small allocations. If a block is bigger than pool size or bigger than page size, then it is allocated outside the pool. If there is not enough memory for small allocation inside pool, then a new block of the same pool size is allocated. This directive has only a very small effect. (source http://markmail.org/message/b2kmrluscevimpba)

default_type

Syntax:	**default_type** *mime-type*
Default:	*text/plain*
Context:	http server location
Reference:	default_type (http://nginx.org/en/docs/http/ngx_http_core_module.html#default_type)

Assigns the default MIME-type to be used for files where the standard MIME map doesn't specify anything.

See also types

directio

Syntax:	**directio** *size* | `off`
Default:	*off*
Context:	http server location
Appeared in:	0.7.7
Reference:	directio (http://nginx.org/en/docs/http/ngx_http_core_module.html#directio)

The directive enables use of flags O_DIRECT (FreeBSD, Linux), F_NOCACHE (Mac OS X) or directio() function (Solaris) for reading files with size greater than specified. This directive disables use of sendfile for this request. This directive may be useful for big files:

```
    directio  4m;
```

directio_alignment

Syntax:	**directio_alignment** *size*
Default:	*512*
Context:	http server location
Appeared	0.8.11

in:	
Reference:	directio_alignment (http://nginx.org/en/docs/http/ngx_http_core_module.html#directio_alignment)

disable_symlinks

Syntax:	**disable_symlinks** `off` **disable_symlinks** `on` | `if_not_owner` [`from` = *part*]
Default:	*off*
Context:	http server location
Appeared in:	1.1.15
Reference:	disable_symlinks (http://nginx.org/en/docs/http/ngx_http_core_module.html#disable_symlinks)

error_page

Syntax:	**error_page** *code* ... [= [*response*]] *uri*
Default:	
Context:	http server location if in location
Reference:	error_page (http://nginx.org/en/docs/http/ngx_http_core_module.html#error_page)

The directive specifies the URI that will be shown for the errors indicated.

Example:

```
error_page   404          /404.html;
error_page   502 503 504  /50x.html;
error_page   403          http://example.com/forbidden.html;
error_page   404          = @fetch;
```

Furthermore, it is possible to change the status code of the answer to another, for example:

```
error_page 404 =200 /empty.gif;
error_page 404 =403 /forbidden.gif;
```

Additionally you can have your designated error handler determine the returned status code by using = without specifying a status code.

```
error_page   404 = /404.php;
```

If there is no need to change URI during redirection it is possible to redirect processing of error pages into a named location:

```
location / (
    error_page 404 @fallback;
)

location @fallback (
    proxy_pass http://backend;
)
```

If you want to use error_page for proxy (upstream) status codes, please see http://wiki.nginx.org/HttpProxyModule#proxy_intercept_errors

if_modified_since

Syntax:	**if_modified_since** `off | exact | before`
Default:	*exact*
Context:	http server location
Appeared in:	0.7.24
Reference:	if_modified_since (http://nginx.org/en/docs/http/ngx_http_core_module.html#if_modified_since)

Specifies how to compare time of file modification and time in request header "If-Modified-Since":

- off — don't check "If-Modified-Since" request header (0.7.34);
- exact — exact match;
- before — file modification time should be less than time in "If-Modified-Since" request header.

ignore_invalid_headers

Syntax:	**ignore_invalid_headers** `on | off`
Default:	*on*
Context:	http server
Reference:	ignore_invalid_headers (http://nginx.org/en/docs/http/ngx_http_core_module.html#ignore_invalid_headers)

internal

Syntax:	**internal**
Default:	
Context:	location

Reference:	internal (http://nginx.org/en/docs/http/ngx_http_core_module.html#internal)

internal indicates that the matching location can be used only for so called "internal" requests.

For external requests it will return the error "Not found" (404).

Internal requests are the following:

- requests redirected by the instruction **error_page**
- subrequests created by the command **include virtual** of the "ngx_http_ssi_module" module
- requests changed by the instruction **rewrite** of the "ngx_http_rewrite_module" module

An example to prevent clients fetching error pages directly:

```
error_page 404 /404.html;
location  /404.html {
  internal;
}
```

keepalive_disable

Syntax:	**keepalive_disable** `none` | *browser ...*
Default:	*msie6*
Context:	http server location
Reference:	keepalive_disable (http://nginx.org/en/docs/http/ngx_http_core_module.html#keepalive_disable)

Disable keepalive for certain user agents (0.9.0+). By default keepalive is disabled for MS Internet Explorer (older than 6.0 service pack 2) after `POST` requests, and for Safari. This is because both browsers have issues with handling `POST` requests with keepalives. If you are running a site that does not use `POST` anywhere, you may optionally choose to enable keepalive in these browsers.

keepalive_timeout

Syntax:	**keepalive_timeout** *timeout* [*header_timeout*]
Default:	*75s*
Context:	http server location
Reference:	keepalive_timeout (http://nginx.org/en/docs/http/ngx_http_core_module.html#keepalive_timeout)

The first parameter assigns the timeout for keep-alive connections with the client. The server will close connections after this time.

The optional second parameter assigns the `time` value in the header `Keep-Alive: timeout=time` of the response. This header can convince some browsers to close the connection, so that the server does not have to. Without this parameter, nginx does not send a `Keep-Alive` header (though this is not what makes a connection "keep-alive").

The parameters can differ from each other.

Notes on how browsers handle the `Keep-Alive` header:

- MSIE and Opera ignore the "Keep-Alive: timeout=<N>" header.
- MSIE keeps the connection alive for about 60-65 seconds, then sends a TCP RST.
- Opera keeps the connection alive for a long time.
- Mozilla keeps the connection alive for N plus about 1-10 seconds.
- Konqueror keeps the connection alive for about N seconds.

keepalive_requests

Syntax:	**keepalive_requests** *number*
Default:	*100*
Context:	http server location
Appeared in:	0.8.0
Reference:	keepalive_requests (http://nginx.org/en/docs/http/ngx_http_core_module.html#keepalive_requests)

Number of requests which can be made over a keep-alive connection.

large_client_header_buffers

Syntax:	**large_client_header_buffers** *number size*
Default:	*4 8k*
Context:	http server
Reference:	large_client_header_buffers (http://nginx.org/en/docs/http/ngx_http_core_module.html#large_client_header_buffers)

Directive assigns the maximum number and size of buffers for large headers to read from client request.

The request line can not be bigger than the size of one buffer, if the client send a bigger header nginx returns error "Request URI too large" (414).

The longest header line of request also must be not more than the size of one buffer, otherwise the client get the error "Bad request" (400).

Buffers are separated only as needed.

By default the size of one buffer is 8192 bytes. In the old nginx, this is equal to the size of page, depending on platform this either 4K or 8K, if at the end of working request connection converts to state keep-alive, then these buffers are freed.

limit_except

Syntax:	**limit_except** *method* ... { ... }
Default:	
Context:	location
Reference:	limit_except (http://nginx.org/en/docs/http/ngx_http_core_module.html#limit_except)

Limits which HTTP methods are allowed for a given request path/location.

For the limitation can be used the directives of modules ngx_http_access_module and ngx_http_auth_basic_module:

```
limit_except  GET {
  allow  192.168.1.0/32;
  deny   all;
}
```

limit_rate

Syntax:	**limit_rate** *rate*
Default:	*0*
Context:	http server location if in location
Reference:	limit_rate (http://nginx.org/en/docs/http/ngx_http_core_module.html#limit_rate)

Directive assigns the speed of transmission of the answer to client. Speed is assigned in the bytes per second. Limitation works only for one connection, i.e., if client opens 2 connections, then total velocity will be 2 times higher then the limit set.

If it is necessary to limit speed for the part of the clients at the *server* level, based on some kind of condition - then this directive does not apply. Instead you should specify the limit by assigning the value to the $limit_rate variable, as shown below:

```
server {
  if ($slow) {
    set $limit_rate  4k;
  }
}
```

You can also control the rate of individual responses returned by a `proxy_pass` response (HttpProxyModule) by setting the `X-Accel-Limit-Rate` header (XSendfile). This can be done without a `X-Accel-Redirect` header.

limit_rate_after

Syntax:	**limit_rate_after** *size*
Default:	*0*
Context:	http server location if in location
Appeared in:	0.8.0
Reference:	limit_rate_after (http://nginx.org/en/docs/http/ngx_http_core_module.html#limit_rate_after)

The directive limits speed only after the first part was sent.

```
limit_rate_after 1m;
limit_rate 100k;
```

lingering_close

Syntax:	**lingering_close** `off` | `on` | `always`
Default:	*on*
Context:	http server location
Appeared in:	1.1.0 1.0.6
Reference:	lingering_close (http://nginx.org/en/docs/http/ngx_http_core_module.html#lingering_close)

Sets SO_LINGER on sockets.

lingering_time

Syntax:	**lingering_time** *time*
Default:	*30s*
Context:	http server location
Reference:	lingering_time (http://nginx.org/en/docs/http/ngx_http_core_module.html#lingering_time)

Sets SO_LINGER on sockets.

lingering_timeout

Syntax:	**lingering_timeout** *time*

Default:	*5s*
Context:	http server location
Reference:	lingering_timeout (http://nginx.org/en/docs/http/ngx_http_core_module.html#lingering_timeout)

Sets SO_LINGER on sockets.

listen

Syntax:	**listen** *address* [: *port*] [`default_server`] [`setfib` = *number*] [`backlog` = *number*] [`rcvbuf` = *size*] [`sndbuf` = *size*] [`accept_filter` = *filter*] [`deferred`] [`bind`] [`ipv6only` = `on` | `off`] [`ssl`] [`so_keepalive` = `on` | `off` |[*keepidle*]:[*keepintvl*]:[*keepcnt*]] **listen** *port* [`default_server`] [`setfib` = *number*] [`backlog` = *number*] [`rcvbuf` = *size*] [`sndbuf` = *size*] [`accept_filter` = *filter*] [`deferred`] [`bind`] [`ipv6only` = `on` | `off`] [`ssl`] [`so_keepalive` = `on` | `off` |[*keepidle*]:[*keepintvl*]:[*keepcnt*]] **listen** `unix:` *path* [`default_server`] [`backlog` = *number*] [`rcvbuf` = *size*] [`sndbuf` = *size*] [`accept_filter` = *filter*] [`deferred`] [`bind`] [`ssl`] [`so_keepalive` = `on` | `off` |[*keepidle*]:[*keepintvl*]:[*keepcnt*]]
Default:	**:80 | *:8000*
Context:	server
Reference:	listen (http://nginx.org/en/docs/http/ngx_http_core_module.html#listen)

The *listen* directive specifies the address and port accepted by the enclosing server {...} block. It is possible to specify only an address, only a port, or a server name as the address.

```
listen 127.0.0.1:8000;
listen 127.0.0.1;
listen 8000;
listen *:8000;
listen localhost:8000;
```

IPv6 address(0.7.36) are set in square brackets:

```
listen [::]:8000;
listen [fe80::1];
```

In Linux by default any IPv6 TCP socket **also accepts** IPv4 traffic using the IPv4 to IPv6 mapped address format, i.e., ::ffff:<IPv4 adddress in dotted decimal notation>. E.g., `::ffff:192.168.0.27` maps the IPv4 address 192.168.0.27 to an IPv6 address.

When you enable the address [::]:80, binding port 80 using IPv6, in the listen directive, in Linux, by default, the IPv4 port 80 is also enabled. Meaning that nginx listens for **both** IPv4 and IPv6 incoming traffic. Therefore if you erroneously specify also a IPv4 address you'll get an *already bind address* error when reloading nginx configuration.

In Linux the separation of the IPv6 and IPv4 stacks is controlled through the runtime parameter: `net.ipv6.bindv6only` which has the value 0 by default.

If you want to use **separate** sockets for IPv4 and IPv6 you should set this parameter to 1 using `sysctl`.

Note that any nginx instance that was running **before** you made the change will **continue to accept** IPv4 traffic. Therefore you should edit your nginx configuration to reflect the new setup for IPv6 and IPv4 packet handling and do a **restart**.

If on the other hand you launched another server instance (vhost) and you expect it to also handle IPv4 traffic by using only, for example:

```
listen [::]:80;
```

the binding of the IPv4 address will fail. The correct way to to this is by using the "ipv6only=on" option in the IPv6 listen directive and also specifying a IPv4 listen directive in the respective server block.

This re-editing of the configuration must be done **after** you changed your kernel runtime parameter. This is the most generic situation in that case (separation of IPv6 and IPv4 sockets):

```
listen [::]:80 ipv6only=on; # listen for IPv6 only traffic c
listen 80; # listen also for IPv4 traffic on "regular" IPv4
```

In FreeBSD the default is **separate** IPv4 and IPv6 sockets. Therefore "listen [::]:80" only binds port 80 for listening to IPv6 traffic. It's always necessary to specify also IPv4 listen directives if you wish to also handle IPv4 traffic.

It's possible to specify **only** IPv6 addresses in the listen directive. Using the "default_server ipv6only=on" option. Specific IPv6 addresses can be used with a IPv6 only default directive. Other server directives can also specifiy listen directives with IPv4 addresses. The uniqueness of the IPv6 handling concerns only the **same** server {...} block.

```
listen [2a02:750:5::123]:80;
listen [::]:80 default_server ipv6only=on;
```

If only the address is given, the default port nginx binds to is 80.

If the directive has the *default_server* parameter, then the enclosing server {...} block will be the default server for the address:port pair. This is useful for name-based virtual hosting where you wish to specify the default server block for hostnames that do not match any server_name directives. If there are no directives with the *default_server* parameter, then the default server will be the first server block in which the `address:port` pair appears. The *default_server* parameter appeared in version 0.8.21 thus deprecating the parameter *default*.

The `listen` directive accepts several parameters, specific to the system calls `listen(2)` and `bind(2)`. These parameters must follow the `default` parameter.

backlog=num -- is assigned parameter backlog in call `listen(2)`. By default backlog equals -1.

rcvbuf=size -- assigned to the parameter `SO_RCVBUF` for the listening socket.

sndbuf=size -- assigned to the parameter `SO_SNDBUF` for the listening socket.

accept_filter=filter -- is assigned name accept-filter.

> . It works only to FreeBSD, it is possible to use two filters -- `dataready` and `httpready`. On the signal -HUP accept-filter it is possible to change only in the quite last versions FreeBSD: 6.0, 5.4-STABLE and 4.11-STABLE.

deferred -- indicates to use that postponed accept(2) on Linux with

> . the aid of option `TCP_DEFER_ACCEPT`.

bind -- indicates that it is necessary to make `bind(2)` separately

> . for this pair of address:port. The fact is that if are described several directives listen with the identical port, but by different addresses and one of the directives listen listens to on all addresses for this port (*:port), then nginx will make bind(2) only to *:port. It is necessary to consider that in this case for determining the address, on which the connections arrive, is done the system call getsockname(). But if are used parameters backlog, rcvbuf, sndbuf, accept_filter or deferred, then it is always done separately for this pair of address:port bind(2).

ssl -- parameter (0.7.14) not related to listen(2) and bind(2) syscalls

> . but instead specifies that connections accepted on this port should work in SSL mode. This allows to specify compact configurations for servers working with both HTTP and HTTPS. For example:

```
listen  80;
listen  443 default_server ssl;
```

Example of the use of the parameters:

```
listen  127.0.0.1 default_server accept_filter=dataready bac
```

Since version 0.8.21 nginx is able to listen on unix sockets:

```
listen unix:/tmp/nginx1.sock;
```

location

Syntax:	**location** [= | ~ | ~* | ^~] *uri* { ... }
	location { } @ *name* { ... }

Default:	
Context:	server location
Reference:	location (http://nginx.org/en/docs/http/ngx_http_core_module.html#location)

This directive allows different configurations depending on the URI. It can be configured using both literal strings and regular expressions. To use regular expressions, you must use a prefix:

1. "~" for case sensitive matching
2. "~*" for case insensitive matching
3. there is no syntax for NOT matching a regular expression. Instead, match the target regular expression and assign an empty block, then use *location* / to match

anything else.

The order in which *location* directives are checked is as follows:

1. Directives with the "=" prefix that match the query exactly (literal string). If found, searching stops.
2. All remaining directives with conventional strings. If this match used the "^~" prefix, searching stops.
3. Regular expressions, in the order they are defined in the configuration file.
4. If #3 yielded a match, that result is used. Otherwise, the match from #2 is used.

Details below.

To determine which *location* directive matches a particular query, the literal strings are checked first. Literal strings match the beginning portion of the query - the most specific match will be used. Afterwards, regular expressions are checked in the order defined in the configuration file. The first regular expression to match the query will stop the search. If no regular expression matches are found, the result from the literal string search is used.

For case-insensitive operating systems, like Mac OS X or Windows with Cygwin, literal string matching is done in a case insensitive way (0.7.7). However, comparison is limited to single-byte locale's only.

Regular expression may contain captures (0.7.40), which can then be used in other directives.

It is possible to disable regular expression checks after literal string matching by using "^~" prefix. If the most specific match literal location has this prefix: regular expressions aren't checked.

The "=" prefix forces an **exact** (literal) match between the request URI and the *location* parameter. When matched, the search stops immediately. A useful application is that if the request "/" occurs frequently, it's better to use "location = /", as that will speed up the processing of this request a bit, since the search will stop after the first comparison.

It is important to know that nginx does the comparison against decoded URIs. For example, if you wish to match "/images/%20/test", then you must use "/images/ /test" to determine the location.

The location directive only tries to match from the first / after the hostname, to just before the first ? or #. (Within that range, it matches the unescaped url.)

Example:

```
location  = / {
  # matches the query / only.
  [ configuration A ]
}
location  / {
  # matches any query, since all queries begin with /, but r
  # expressions and any longer conventional blocks will be
  # matched first.
  [ configuration B ]
}
location /documents/ {
  # matches any query beginning with /documents/ and continu
  # so regular expressions will be checked. This will be mat
  # regular expressions don't find a match.
  [ configuration C ]
}
location ^~ /images/ {
  # matches any query beginning with /images/ and halts sear
  # so regular expressions will not be checked.
  [ configuration D ]
}
location ~* \.(gif|jpg|jpeg)$ {
  # matches any request ending in gif, jpg, or jpeg. Howeve
  # requests to the /images/ directory will be handled by
  # Configuration D.
  [ configuration E ]
}
```

Example requests:

- / -> configuration A
- /index.html -> configuration B
- /documents/document.html -> configuration C
- /images/1.gif -> configuration D
- /documents/1.jpg -> configuration E

Note that you could define these 5 configurations in any order and the results would remain the same.

The prefix "@" specifies a named location. Such locations are not used during normal processing of requests, they are intended only to process internally redirected requests (see error_page, try_files).

log_not_found

Syntax:	**log_not_found** `on	off`
Default:	*on*
Context:	http server location
Reference:	log_not_found (http://nginx.org/en/docs/http/ngx_http_core_module.html#log_not_found)

The directive enables or disables messages in error_log about files not found on disk.

log_subrequest

Syntax:	**log_subrequest** `on	off`
Default:	*off*
Context:	http server location
Reference:	log_subrequest (http://nginx.org/en/docs/http/ngx_http_core_module.html#log_subrequest)

The directive enables or disables messages in access_log about sub-requests such as rewrite rules and/or SSI requests.

max_ranges

Syntax:	**max_ranges** *number*
Default:	
Context:	http server location
Appeared in:	1.1.2
Reference:	max_ranges (http://nginx.org/en/docs/http/ngx_http_core_module.html#max_ranges)

merge_slashes

Syntax:	**merge_slashes** `on` | `off`
Default:	*on*
Context:	http server
Reference:	merge_slashes (http://nginx.org/en/docs/http/ngx_http_core_module.html#merge_slashes)

Enables merging adjacent slashes when parsing the request line. For example, a request for http://www.example.com/foo//bar/ will produce the following values for $uri:

- on: /foo/bar/
- off: /foo//bar/

Be aware that static location matching is performed as a string compare, so if merge_slashes is turned off, a request for /foo//bar/ will *not* match `location /foo/bar/`.

msie_padding

Syntax:	**msie_padding** `on` | `off`
Default:	*on*
Context:	http server location
Reference:	msie_padding (http://nginx.org/en/docs/http/ngx_http_core_module.html#msie_padding)

This directive enables or disables the msie_padding feature for MSIE browsers, and Chrome (as of nginx 0.8.25+). When this is enabled, nginx will pad the size of the response body to a minimum of 512 bytes for responses with a status code above or equal to 400.

The padding prevents the activation of "friendly" HTTP error pages in MSIE and Chrome, so as to not hide/mask the more-informative error pages from the server.

msie_refresh

Syntax:	**msie_refresh** `on` | `off`
Default:	*off*
Context:	http server location
Reference:	msie_refresh (http://nginx.org/en/docs/http/ngx_http_core_module.html#msie_refresh)

This directive allows or forbids issuing a `refresh` instead of doing a `redirect` for MSIE.

open_file_cache

Syntax:	**open_file_cache** `off` **open_file_cache** `max` = *N* [`inactive` = *time*]
Default:	*off*
Context:	http server location
Reference:	open_file_cache (http://nginx.org/en/docs/http/ngx_http_core_module.html#open_file_cache)

The directive sets the cache activity on. These information can be stored:

- Open file descriptors, information with their size and modification time;
- Information about the existence of directories;
- Error information when searches for a file - no file, do not have rights to read, etc. See also open_file_cache_errors

Options directive:

- `max` - specifies the maximum number of entries in the cache. When the cache overflows, the least recently used(LRU) items will be removed;
- `inactive` - specifies the time when the cached item is removed, if it has not been downloaded during that time, the default is 60 seconds;
- `off` - prohibits the cache activity.

Example:

```
open_file_cache max=1000 inactive=20s;
open_file_cache_valid    30s;
open_file_cache_min_uses 2;
open_file_cache_errors   on;
```

open_file_cache_errors

Syntax:	**open_file_cache_errors** `on` | `off`
Default:	*off*
Context:	http server location
Reference:	open_file_cache_errors (http://nginx.org/en/docs/http/ngx_http_core_module.html#open_file_cache_errors)

The directive specifies whether or not to cache errors when searching for a file.

open_file_cache_min_uses

Syntax:	**open_file_cache_min_uses** *number*
Default:	*1*
Context:	http server location
Reference:	open_file_cache_min_uses (http://nginx.org/en/docs/http/ngx_http_core_module.html#open_file_cache_min_uses)

The directive defines the minimum use number of a file within the time specified in the directive parameter inactive in open_file_cache. ?If use more than the number, the file descriptor will remain open in the cache.

open_file_cache_valid

Syntax:	**open_file_cache_valid** *time*
Default:	*60s*
Context:	http server location
Reference:	open_file_cache_valid (http://nginx.org/en/docs/http/ngx_http_core_module.html#open_file_cache_valid)

The directive specifies the time when need to check the validity of the information about the item in open_file_cache.

optimize_server_names

Syntax:	**optimize_server_names** `on` | `off`
Default:	*off*
Context:	http server
Reference:	optimize_server_names (http://nginx.org/en/docs/http/ngx_http_core_module.html#optimize_server_names)

Directive activates or deactivates optimization of host name checks for name-based virtual servers.

In particular, the check influences the name of the host used in redirects. If optimization is on, and all name-based servers listening on one address:port pair have identical configuration, then names are not checked during request execution and redirects use first server name.

If redirect must use host name passed by the client, then the optimization must be turned off.

Note: this directive is deprecated in nginx 0.7.x, use server_name_in_redirect instead.

port_in_redirect

Syntax:	**port_in_redirect** `on` | `off`
Default:	*on*
Context:	http server location
Reference:	port_in_redirect (http://nginx.org/en/docs/http/ngx_http_core_module.html#port_in_redirect)

Directive allows or prevents port indication in redirects handled by nginx.

If `port_in_redirect` is off, then Nginx will **not** add the port in the url when the request is redirected.

post_action

syntax: *post_action [uri|off]*

default: *post_action off*

context: *http, server, location, if-in-location*

Defines a URI to sub-request upon completion of current request.

```
location /protected_files {
        internal;

        proxy_pass http://127.0.0.2;
        post_action /protected_done;
}

# Send the post_action request to a FastCGI backend for logg
location /protected_done {
        internal;
        fastcgi_pass 127.0.0.1:9000;
}
```

Note: this directive "has subtleties" according to Maxim Dounin, so **use at your own risk.**

postpone_output

Syntax:	**postpone_output** *size*
Default:	*1460*
Context:	http server location
Reference:	postpone_output (http://nginx.org/en/docs/http/ngx_http_core_module.html#postpone_output)

read_ahead

Syntax:	**read_ahead** *size*
Default:	*0*
Context:	http server location
Reference:	read_ahead (http://nginx.org/en/docs/http/ngx_http_core_module.html#read_ahead)

recursive_error_pages

Syntax:	**recursive_error_pages** `on` | `off`
Default:	*off*
Context:	http server location
Reference:	recursive_error_pages (http://nginx.org/en/docs/http/ngx_http_core_module.html#recursive_error_pages)

`recursive_error_pages` enables or disables following a chain of `error_page` directives.

request_pool_size

Syntax:	**request_pool_size** *size*
Default:	*4k*
Context:	http server
Reference:	request_pool_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#request_pool_size)

The directive is used to allocate memory per request. The pool is used for small allocations. If a block is bigger than pool size or bigger than page size, then it is allocated outside the pool. If there is not enough memory for small allocation inside pool, then a new block of the same pool size is allocated. This directive has only a very small effect. (source http://markmail.org/message/b2kmrluscevimpba)

reset_timedout_connection

Syntax:	**reset_timedout_connection** `on` | `off`
Default:	*off*
Context:	http server location
Reference:	reset_timedout_connection (http://nginx.org/en/docs/http/ngx_http_core_module.html#reset_timedout_connection)

This directive enables or disables resetting the connection on timeout. When resetting the connection, before the socket is closed, the socket SO_LINGER option is set with a 0 timeout, which forces the RST packet to be sent to the client upon closing the socket, thus freeing all memory associated with it. This prevents the socket in the FIN_WAIT1 state, along with the buffers associated with it from lying around.

Note that sockets with keepalive connections, after the defined timeout, are closed in the usual way.

resolver

Syntax:	**resolver** *address* ... [`valid` = *time*]
Default:	
Context:	http server location
Reference:	resolver (http://nginx.org/en/docs/http/ngx_http_core_module.html#resolver)

Directive defines DNS server address, e.g.

```
resolver 127.0.0.1;
```

resolver_timeout

Syntax:	**resolver_timeout** *time*
Default:	*30s*
Context:	http server location
Reference:	resolver_timeout (http://nginx.org/en/docs/http/ngx_http_core_module.html#resolver_timeout)

Directive defines timeout for name resolution, e.g.

```
resolver_timeout 5s;
```

root

Syntax:	**root** *path*
Default:	*html*
Context:	http server location if in location
Reference:	root (http://nginx.org/en/docs/http/ngx_http_core_module.html#root)

root specifies the document root for the requests. For example, with this configuration

```
location  /i/ {
  root  /spool/w3;
}
```

A request for "/i/top.gif" will return the file "/spool/w3/i/top.gif". You can use variables in the argument.

note: Keep in mind that the root will still append the directory to the request so that a request for "/i/top.gif" will not look in "/spool/w3/top.gif" like might happen in an Apache-like alias configuration where the location match itself is dropped. Use the `alias` directive to achieve the Apache-like functionality.

satisfy

Syntax:	**satisfy** `all` | `any`
Default:	*all*
Context:	http server location
Reference:	satisfy (http://nginx.org/en/docs/http/ngx_http_core_module.html#satisfy)

This determines the adopted access policy when directives from multiple access phase handlers, such as the Access and Auth Basic modules, are defined in a context:

- all - All access phase handlers must grant access to the context
- any - Any access phase handler may grant access to the context

```
location / {
  satisfy any;
  allow 192.168.1.0/32;
  deny all;
  auth_basic "closed site";
  auth_basic_user_file conf/htpasswd;
}
```

satisfy_any

Syntax:	**satisfy_any** `on` | `off`
Default:	*off*
Context:	http server location
Reference:	satisfy_any (http://nginx.org/en/docs/http/ngx_http_core_module.html#satisfy_any)

deprecated: 0.6.25 -- Use the satisfy directive instead

send_lowat

Syntax:	**send_lowat** *size*
Default:	*0*
Context:	http server location
Reference:	send_lowat (http://nginx.org/en/docs/http/ngx_http_core_module.html#send_lowat)

send_timeout

Syntax:	**send_timeout** *time*
Default:	*60s*
Context:	http server location
Reference:	send_timeout (http://nginx.org/en/docs/http/ngx_http_core_module.html#send_timeout)

Specifies the response timeout to the client. This timeout does not apply to the *entire* transfer but, rather, only between two subsequent client-read operations. Thus, if the client has not read any data for this amount of time, then nginx shuts down the connection.

sendfile

Syntax:	**sendfile** `on` | `off`
Default:	*off*
Context:	http server location if in location
	sendfile

Reference:	(http://nginx.org/en/docs/http/ngx_http_core_module.html#sendfile)

Directive activate or deactivate the usage of `sendfile()`.

sendfile() copies data between one file descriptor and another. Because this copying is done within the kernel, sendfile() is more efficient than the combination of read(2) and write(2), which would require transferring data to and from user space.

Read more at: https://www.kernel.org/doc/man-pages/online/pages/man2/sendfile.2.html

How sendfile helps : http://www.techrepublic.com/article/use-sendfile-to-optimize-data-transfer/1044112

sendfile_max_chunk

Syntax:	**sendfile_max_chunk** *size*
Default:	*0*
Context:	http server location
Reference:	sendfile_max_chunk (http://nginx.org/en/docs/http/ngx_http_core_module.html#sendfile_max_chunk)

server

Syntax:	**server** { ... }
Default:	
Context:	http
Reference:	server (http://nginx.org/en/docs/http/ngx_http_core_module.html#server)

Directive assigns configuration for the virtual server.

There is no separation of IP and name-based (the `Host` header of the request) servers.

Instead, the directive `listen` is used to describe all addresses and ports on which incoming connections can occur, and in directive `server_name` indicate all names of the server.

server_name

Syntax:	**server_name** *name ...*
Default:	""
Context:	server
Reference:	server_name (http://nginx.org/en/docs/http/ngx_http_core_module.html#server_name)

This directive performs two actions:

- Compares the `Host` header of the incoming HTTP request against the server { ... } blocks in the Nginx configuration files and selects the first one that matches. This is how **virtual servers** are defined. Server names are processed in the following order:

1. full, static names
2. names with a wildcard at the start of the name — *.example.com
3. names with a wildcard at the end of the name — www.example.*
4. names with regular expressions

   If there is no match, a server { ... } block in the configuration file will be used based on the following order:

1. the server block with a matching `listen` directive marked as `[default|default_server]`
2. the first server block with a matching `listen` directive (or implicit `listen 80;`)

- Sets the server name that will be used in HTTP redirects if server_name_in_redirect is set.

Example:

```
server {
  server_name   example.com  www.example.com;
}
```

The first name becomes the basic name of server. By default the name of the machine (hostname) is used.

It is possible to use "*" for replacing the first or the last part of the name:

```
server {
  server_name   example.com  *.example.com  www.example.*;
}
```

The first two of the above names (example.com and *.example.com) can be combined into one:

```
server {
  server_name  .example.com;
}
```

It is also possible to use regular expressions in server names, prepending the name with a tilde "~" like so:

```
server {
  server_name   www.example.com   ~^www\d+\.example\.com$;
}
```

Since nginx 0.7.12, an empty server name is supported to catch the requests without "Host" header, please note that most browsers will always send a Host header, if accessed by IP the Host header will contain the IP. To specify a catch-all block please see the default_server flag of the listen directive.

```
server {
  server_name "";
}
```

Since nginx 0.8.25 named captures can be used in server_name:

```
server {
  server_name   ~^(www\.)?(?<domain>.+)$;
  root  /sites/$domain;
}
```

and multiple name captures:

```
server {
  server_name   ~^(?<subdomain>.+?)\.(?<domain>.+)$;
  root  /sites/$domain/$subdomain;
}
```

Some older versions of PCRE may have issues with this syntax. If any problems arise try this following syntax:

```
server {
  server_name   ~^(www\.)?(?P<domain>.+)$;
  root  /sites/$domain;
}
```

Since nginx 0.9.4, $hostname can be used as a server_name argument:

```
server {
  server_name $hostname;
}
```

server_name_in_redirect

Syntax:	**server_name_in_redirect** `on` | `off`
Default:	*off*
Context:	http server location
Reference:	server_name_in_redirect (http://nginx.org/en/docs/http/ngx_http_core_module.html#server_name_in_redirect)

If `server_name_in_redirect` is on, then Nginx will use the first value of the server_name directive for redirects. If `server_name_in_redirect` is off, then nginx will use the requested `Host` header.

Note: for Location headers coming from an upstream proxy (via proxy_pass for example) this may not be the only directive you need. In fact, it seems to be ignored a lot of the time. If you are seeing the upstream's server name come through and not be rewritten, you will need to use proxy_redirect to rewrite the upstream's provided hostname to what you want. Something like `proxy_redirect http://some.upstream.url/ /` - you will want to rewrite it to a / relative path.

server_names_hash_max_size

Syntax:	**server_names_hash_max_size** *size*
Default:	*512*
Context:	http
Reference:	server_names_hash_max_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#server_names_hash_max_size)

The maximum size of the server name hash tables. For more detail see the description of tuning the hash tables in Nginx Optimizations.

server_names_hash_bucket_size

Syntax:	**server_names_hash_bucket_size** *size*
Default:	*32/64/128*
Context:	http
Reference:	server_names_hash_bucket_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#server_names_hash_bucket_size)

Directive assigns the size of basket in the hash-tables of the names of servers. This value by default depends on the size of the line of processor cache. For more detail see the description of tuning the hash tables in Nginx Optimizations.

server_tokens

Syntax:	**server_tokens** `on	off`
Default:	*on*
Context:	http server location
Reference:	server_tokens (http://nginx.org/en/docs/http/ngx_http_core_module.html#server_tokens)

Whether to send the Nginx version number in error pages and `Server` header.

tcp_nodelay

Syntax:	**tcp_nodelay** `on	off`
Default:	*on*
Context:	http server location
Reference:	tcp_nodelay (http://nginx.org/en/docs/http/ngx_http_core_module.html#tcp_nodelay)

This directive allows or forbids the use of the socket option `TCP_NODELAY`. Only included in `keep-alive` connections.

You can read more about the `TCP_NODELAY` socket option here.

tcp_nopush

Syntax:	**tcp_nopush** `on` | `off`
Default:	*off*
Context:	http server location
Reference:	tcp_nopush (http://nginx.org/en/docs/http/ngx_http_core_module.html#tcp_nopush)

This directive permits or forbids the use of the socket options `TCP_NOPUSH` on FreeBSD or `TCP_CORK` on Linux. This option is only available when using `sendfile`.

Setting this option causes nginx to attempt to send it's HTTP response headers in one packet on Linux and FreeBSD 4.x

You can read more about the `TCP_NOPUSH` and `TCP_CORK` socket options here.

try_files

Syntax:	**try_files** *file ... uri* **try_files** *file ...* = *code*
Default:	
Context:	server location
Reference:	try_files (http://nginx.org/en/docs/http/ngx_http_core_module.html#try_files)

Checks for the existence of files in order, and returns the first file that is found. A trailing slash indicates a directory - `$uri /`. In the event that no file is found, an internal redirect to the last parameter is invoked. Do note that only the last parameter causes an internal redirect, former ones just sets the internal URI pointer. The last parameter is the fallback URI and *must* exist, or else an internal error will be raised. Named locations can be used. Unlike with rewrite, $args are not automatically preserved if the fallback is not a named location. If you need args preserved, you must do so explicitly:

```
try_files $uri $uri/ /index.php?q=$uri&$args;
```

Example use in proxying Mongrel:

```
try_files /system/maintenance.html $uri $uri/index.html $uri

location @mongrel {
  proxy_pass http://mongrel;
}
```

Note that you can specify an HTTP **status** code as the last argument to `try_file` since Nginx version 0.7.51. Here's an example:

```
location / {
  try_files $uri $uri/ /error.php?c=404 =404;
}
```

When all other attempts to serve the content corresponding to the request fail issue a `404 Not Found`.

Example of use with Drupal / FastCGI:

```
# for Drupal 6 or 7:
try_files $uri $uri/ /index.php?q=$uri&$args;

# a better version for Drupal 7 since it doesn't need q=$uri
try_files $uri $uri/ /index.php?$args;

location ~ \.php$ {
  fastcgi_pass 127.0.0.1:8888;
  fastcgi_param  SCRIPT_FILENAME $document_root$fastcgi_scri
  # any other specific fastcgi_params
}
```

In this example, the directive `try_files`

```
try_files $uri $uri/ /index.php?q=$uri&$args;
```

Is basically the same as this:

```
location / {
  error_page     404 = @drupal;
  log_not_found  off;
}

location @drupal {
  rewrite ^ /index.php?q=$uri last; # for drupal 6
}
```

Or this:

```
# DO NOT DO THIS! This is a terrible use of if.
if (!-e $request_filename) {
   rewrite ^ /index.php?q=$uri last;
}
```

`try_files` is basically a replacement for the typical mod_rewrite style file/directory existence check. It is supposed to be more efficient than using `if` - see IfIsEvil

Examples of use with Wordpress and Joomla (typical "Front controller pattern" packages)

```
# wordpress (without WP Super Cache) - example 1
try_files $uri $uri/ /index.php?q=$uri&$args;

# wordpress (without WP Super Cache) - example 2
# It doesn't REALLY need the "q" parameter, but without an e
# gets an empty QUERY_STRING, breaking generated responses t
# permalink, such as search results.
try_files $uri $uri/ /index.php?$args;
```

```
# joomla
try_files $uri $uri/ /index.php?q=$uri&$args;

location ~ \.php$ {
  fastcgi_pass 127.0.0.1:8888;
  fastcgi_param  SCRIPT_FILENAME $document_root$fastcgi_scri
  # any other specific fastcgi_params
}
```

WP Super Cache requires a bunch of static file checks. Those are not shown here.

types

Syntax:	**types** { ... }
Default:	*{ text/html html; image/gif gif; image/jpeg jpg; }*
Context:	http server location
Reference:	types (http://nginx.org/en/docs/http/ngx_http_core_module.html#types)

Specifies one or more mappings between MIME types and file extensions. More than one extension can be assigned to a MIME type.

For example:

```
types {
  text/html    html;
  image/gif    gif;
  image/jpeg   jpg;
}
```

A sufficiently complete table of mappings is included with nginx, and is located at `conf/mime.types`.

If you wanted responses to particular location to always indicate a single MIME type, you could define an empty *types* block and set the *default_type* directive. For example:

```
location /download/ {
  types         { }
  default_type  application/octet-stream;
}
```

types_hash_bucket_size

Syntax:	**types_hash_bucket_size** *size*
Default:	*32/64/128*
Context:	http server location
Reference:	types_hash_bucket_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#types_hash_bucket_size)

types_hash_max_size

Syntax:	**types_hash_max_size** *size*
Default:	*1024*
Context:	http server location
Reference:	types_hash_max_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#types_hash_max_size)

underscores_in_headers

Syntax:	**underscores_in_headers** `on|off`
Default:	*off*
Context:	http server
Reference:	underscores_in_headers (http://nginx.org/en/docs/http/ngx_http_core_module.html#underscores_in_headers)

Allows or disallows underscores in headers.

variables_hash_bucket_size

Syntax:	**variables_hash_bucket_size** *size*
Default:	*64*
Context:	http
Reference:	variables_hash_bucket_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#variables_hash_bucket_size)

Assigns the key bucket size for the variables hash table.

variables_hash_max_size

Syntax:	**variables_hash_max_size** *size*
Default:	*512*
Context:	http
Reference:	variables_hash_max_size (http://nginx.org/en/docs/http/ngx_http_core_module.html#variables_hash_max_size)

The maximum size of the variables hash table. For more detail see the description of tuning the hash tables in Nginx Optimizations.

Variables

The core module supports built-in variables, whose names correspond with the names of variables in Apache.

First of all, there are variables which represent header lines in the client request, for example, `$http_user_agent`, `$http_cookie`, and so forth. Note that because these correspond to what the client actually sends, they are not guaranteed to exist and their meaning is defined elsewhere (e.g. in relevant standards).

Furthermore, there are other variables:

$arg_PARAMETER

This variable contains the value of the `GET` request variable *PARAMETER* if present in the query string

$args

This variable is the `GET` parameters in request line, e.g. `foo=123&bar=blahblah`; This variable could be changed.

$binary_remote_addr

The address of the client in binary form;

$body_bytes_sent

The amount of bytes sent as part of the body of the response. Is accurate even when connections are aborted or interrupted.

$content_length

This variable is equal to line `Content-Length` in the header of request;

$content_type

This variable is equal to line `Content-Type` in the header of request;

$cookie_COOKIE

The value of the cookie *COOKIE*;

$document_root

This variable is equal to the value of directive root for the current request;

$document_uri

The same as $uri.

$host

This variable is equal to line `Host` in the header of request or name of the server processing the request if the `Host` header is not available.

This variable may have a different value from `$http_host` in such cases: 1) when the `Host` input header is absent or has an empty value, `$host` equals to the value of `server_name` directive; 2)when the value of `Host` contains port number, `$host` doesn't include that port number. `$host`'s value is always lowercase since 0.8.17.

$hostname

Set to the machine's hostname as returned by gethostname (http://linux.die.net/man/2/gethostname)

$http_HEADER

The value of the HTTP request header *HEADER* when converted to lowercase and with 'dashes' converted to 'underscores', e.g. `$http_user_agent`, `$http_referer`...;

$is_args

Evaluates to "?" if $args is set, "" otherwise.

$limit_rate

This variable allows limiting the connection rate.

$nginx_version

The version of Nginx that the server is currently running;

$query_string

The same as $args except that this variable is readonly.

$remote_addr

The address of the client.

$remote_port

The port of the client;

$remote_user

This variable is equal to the name of user, authenticated by the Auth Basic Module;

$request_filename

This variable is equal to path to the file for the current request, formed from directives root or alias and URI request;

$request_body

This variable(0.7.58+) contains the body of the request. The significance of this variable appears in locations with directives proxy_pass or fastcgi_pass.

$request_body_file

Client request body temporary filename;

$request_completion

Set to "OK" if request was completed successfully. Empty if request was not completed or if request was not the last part of a series of range requests.

$request_method

This variable is equal to the method of request, usually `GET` or `POST`.

This variable always evaluates to the method name of the *main request*, not the current request, when the current request is a subrequest.

$request_uri

This variable is equal to the *original* request URI as received from the client including the args. It cannot be modified. Look at $uri for the post-rewrite/altered URI. Does not include host name. Example: "/foo/bar.php?arg=baz"

$scheme

The HTTP scheme (i.e. http, https). Evaluated only on demand, for example:

```
rewrite  ^  $scheme://example.com$uri  redirect;
```

$sent_http_HEADER

The value of the HTTP response header *HEADER* when converted to lowercase and with 'dashes' converted to 'underscores', e.g. `$sent_http_cache_control`, `$sent_http_content_type`... .

$server_addr

The server address. Generally nginx makes a system call to obtain this value. To improve efficiency and avoid this system call, specify an address with the listen directive and to use the `bind` parameter.

$server_name

The name of the server.

$server_port

This variable is equal to the port of the server, to which the request arrived;

$server_protocol

This variable is the protocol of the request. Common examples are: `HTTP/1.0` or `HTTP/1.1`

$uri

This variable is the current request URI, without any arguments (see $args for those). This variable will reflect any modifications done so far by internal redirects or the index module. Note this may be different from $request_uri, as $request_uri is what was originally sent by the browser before any such modifications. Does not include the protocol or host name. Example: `/foo/bar.html`

References

Original Documentation (http://nginx.org/en/docs/http/ngx_http_core_module.html)