Modules reference

NGINX Plus - release 3, based on 1.5.12 core

March 21, 2014
Copyright Notice

© 2012-2014 Nginx, Inc. All rights reserved. NGINX, NGINX Plus and any Nginx, Inc. product or service name or logo used herein are trademarks of Nginx, Inc. All other trademarks used herein belong to their respective owners. The trademarks and logos displayed herein may not be used without the prior written consent of Nginx, Inc. or their respective owners.

This documentation is provided “AS IS” and is subject to change without notice and should not be interpreted as a commitment by Nginx, Inc. This documentation may not be copied, modified or distributed without authorization of Nginx, Inc. and may be used only in connection with Nginx, Inc. products and services. Nginx, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this documentation.
Preface

About NGINX

NGINX® (“engine x”) is a high performance, high concurrency web server excelling at large scale content delivery, web acceleration and protecting application containers. Its precise integration with modern operating systems allows unprecedented levels of efficiency even when running on commodity hardware.

Nginx, Inc. develops and maintains NGINX open source distribution, and offers commercial support and professional services for NGINX.

About NGINX Plus

- Offers additional features on top of the free open source NGINX version.
- Prepared, tested and supported by NGINX core engineering team led by the original author Igor Sysoev.

For more information

- Find more details about NGINX products and support at http://nginx.com.
- For online NGINX documentation visit http://nginx.org/en/docs.
- For general inquiries, please use: nginx-inquiries@nginx.com
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>1</td>
</tr>
<tr>
<td>Preface</td>
<td>2</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>3</td>
</tr>
</tbody>
</table>

### 1 Core modules

1.1 Core functionality   18

1.1.1 Example Configuration    18

1.1.2 Directives
- accept_mutex    18
- accept_mutex_delay    18
- daemon    19
- debug_connection    19
- debug_points    19
- error_log    20
- env    21
- events    21
- include    21
- lock_file    22
- master_process    22
- multi_accept    22
- pcre_jit    22
- pid    23
- ssl_engine    23
- timer_resolution    23
- use    24
- user    24
- worker_aio_requests    24
- worker_connections    24
- worker_cpu_affinity    24
- worker_priority    25
- worker_processes    25
- worker_rlimit_core    26
- worker_rlimit_nofile    26
- worker_rlimit_sigpending    26
- working_directory    26
1.2 Setting up hashes ........................................... 27
  1.2.1 Overview ........................................... 27
1.3 Connection processing methods ............................... 28
  1.3.1 Overview ........................................... 28

2 HTTP server modules ...................................... 29
  2.1 Module ngx_http_core_module ............................ 29
    2.1.1 Directives ....................................... 29
      aio ........................................... 29
      alias ........................................ 30
      chunked_transfer_encoding ....................... 31
      client_body_buffer_size ......................... 31
      client_body_in_file_only ......................... 32
      client_body_in_single_buffer ...................... 32
      client_body_temp_path .............................. 32
      client_body_timeout ............................... 32
      client_header_buffer_size ......................... 33
      client_header_timeout .............................. 33
      client_max_body_size ................................ 33
      connection_pool_size ................................ 33
      default_type ...................................... 34
      directio ........................................ 34
      directio_alignment ................................ 34
      disable_symlinks ................................ 34
      error_page ........................................ 35
      etag ............................................. 36
      http ............................................. 36
      if_modified_since ................................ 37
      ignore_invalid_headers ............................. 37
      internal .......................................... 37
      keepalive_disable ................................ 38
      keepalive_requests ................................ 38
      keepalive_timeout ................................ 39
      large_client_header_buffers ....................... 39
      limit_except ...................................... 39
      limit_rate ........................................ 40
      limit_rate_after .................................. 40
      lingering_close .................................. 40
      lingering_time .................................... 41
      lingering_timeout ................................ 41
      listen ............................................. 41
      location .......................................... 44
      log_not_found ..................................... 46
      log_subrequest .................................... 46
      max_ranges ........................................ 46
      merge_slashes ...................................... 46
<table>
<thead>
<tr>
<th>Directive</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>msie_padding</td>
<td>47</td>
</tr>
<tr>
<td>msie_refresh</td>
<td>47</td>
</tr>
<tr>
<td>open_file_cache</td>
<td>47</td>
</tr>
<tr>
<td>open_file_cache_errors</td>
<td>48</td>
</tr>
<tr>
<td>open_file_cache_min_uses</td>
<td>48</td>
</tr>
<tr>
<td>open_file_cache_valid</td>
<td>48</td>
</tr>
<tr>
<td>optimize_server_names</td>
<td>48</td>
</tr>
<tr>
<td>output_buffers</td>
<td>48</td>
</tr>
<tr>
<td>port_in_redirect</td>
<td>49</td>
</tr>
<tr>
<td>postpone_output</td>
<td>49</td>
</tr>
<tr>
<td>read-ahead</td>
<td>49</td>
</tr>
<tr>
<td>recursive_error_pages</td>
<td>49</td>
</tr>
<tr>
<td>request_pool_size</td>
<td>49</td>
</tr>
<tr>
<td>reset_timedout_connection</td>
<td>50</td>
</tr>
<tr>
<td>resolver</td>
<td>50</td>
</tr>
<tr>
<td>resolver_timeout</td>
<td>51</td>
</tr>
<tr>
<td>root</td>
<td>51</td>
</tr>
<tr>
<td>satisfy</td>
<td>51</td>
</tr>
<tr>
<td>satisfy_any</td>
<td>52</td>
</tr>
<tr>
<td>send_lowat</td>
<td>52</td>
</tr>
<tr>
<td>send_timeout</td>
<td>52</td>
</tr>
<tr>
<td>sendfile</td>
<td>52</td>
</tr>
<tr>
<td>sendfile_max_chunk</td>
<td>52</td>
</tr>
<tr>
<td>server</td>
<td>53</td>
</tr>
<tr>
<td>server_name</td>
<td>53</td>
</tr>
<tr>
<td>server_name_in_redirect</td>
<td>55</td>
</tr>
<tr>
<td>server_names_hash_bucket_size</td>
<td>55</td>
</tr>
<tr>
<td>server_names_hash_max_size</td>
<td>55</td>
</tr>
<tr>
<td>server_tokens</td>
<td>55</td>
</tr>
<tr>
<td>tcp_nodelay</td>
<td>55</td>
</tr>
<tr>
<td>tcp_nopush</td>
<td>56</td>
</tr>
<tr>
<td>try_files</td>
<td>56</td>
</tr>
<tr>
<td>types</td>
<td>58</td>
</tr>
<tr>
<td>types_hash_bucket_size</td>
<td>58</td>
</tr>
<tr>
<td>types_hash_max_size</td>
<td>59</td>
</tr>
<tr>
<td>underscores_in_headers</td>
<td>59</td>
</tr>
<tr>
<td>variables_hash_bucket_size</td>
<td>59</td>
</tr>
<tr>
<td>variables_hash_max_size</td>
<td>59</td>
</tr>
<tr>
<td>2.1.2 Embedded Variables</td>
<td>59</td>
</tr>
<tr>
<td>2.2 Module ngx_http_access_module</td>
<td>63</td>
</tr>
<tr>
<td>2.2.1 Summary</td>
<td>63</td>
</tr>
<tr>
<td>2.2.2 Example Configuration</td>
<td>63</td>
</tr>
<tr>
<td>2.2.3 Directives</td>
<td>63</td>
</tr>
<tr>
<td>allow</td>
<td>63</td>
</tr>
<tr>
<td>deny</td>
<td>63</td>
</tr>
<tr>
<td>2.3 Module ngx_http_addition_module</td>
<td>64</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>dav_access</td>
<td>75</td>
</tr>
<tr>
<td>dav_methods</td>
<td>76</td>
</tr>
<tr>
<td>create_full_put_path</td>
<td>76</td>
</tr>
<tr>
<td>min_delete_depth</td>
<td>76</td>
</tr>
<tr>
<td>2.10 Module ngx_http_empty_gif_module</td>
<td>77</td>
</tr>
<tr>
<td>2.10.1 Summary</td>
<td>77</td>
</tr>
<tr>
<td>2.10.2 Example Configuration</td>
<td>77</td>
</tr>
<tr>
<td>2.10.3 Directives</td>
<td>77</td>
</tr>
<tr>
<td>empty_gif</td>
<td>77</td>
</tr>
<tr>
<td>2.11 Module ngx_http_fastcgi_module</td>
<td>78</td>
</tr>
<tr>
<td>2.11.1 Summary</td>
<td>78</td>
</tr>
<tr>
<td>2.11.2 Example Configuration</td>
<td>78</td>
</tr>
<tr>
<td>2.11.3 Directives</td>
<td>78</td>
</tr>
<tr>
<td>fastcgi_bind</td>
<td>78</td>
</tr>
<tr>
<td>fastcgi_buffer_size</td>
<td>78</td>
</tr>
<tr>
<td>fastcgi_buffering</td>
<td>79</td>
</tr>
<tr>
<td>fastcgi_buffers</td>
<td>79</td>
</tr>
<tr>
<td>fastcgi_busy_buffers_size</td>
<td>79</td>
</tr>
<tr>
<td>fastcgi_cache</td>
<td>79</td>
</tr>
<tr>
<td>fastcgi_cache_bypass</td>
<td>80</td>
</tr>
<tr>
<td>fastcgi_cache_key</td>
<td>80</td>
</tr>
<tr>
<td>fastcgi_cache_lock</td>
<td>80</td>
</tr>
<tr>
<td>fastcgi_cache_lock_timeout</td>
<td>80</td>
</tr>
<tr>
<td>fastcgi_cache_methods</td>
<td>81</td>
</tr>
<tr>
<td>fastcgi_cache_min_uses</td>
<td>81</td>
</tr>
<tr>
<td>fastcgi_cache_path</td>
<td>81</td>
</tr>
<tr>
<td>fastcgi_cache_purge</td>
<td>82</td>
</tr>
<tr>
<td>fastcgi_cache_revalidate</td>
<td>83</td>
</tr>
<tr>
<td>fastcgi_cache_use_stale</td>
<td>83</td>
</tr>
<tr>
<td>fastcgi_cache_valid</td>
<td>83</td>
</tr>
<tr>
<td>fastcgi_catch_stderr</td>
<td>84</td>
</tr>
<tr>
<td>fastcgi_connect_timeout</td>
<td>84</td>
</tr>
<tr>
<td>fastcgi_hide_header</td>
<td>84</td>
</tr>
<tr>
<td>fastcgi_ignore_client_abort</td>
<td>85</td>
</tr>
<tr>
<td>fastcgi_ignore_headers</td>
<td>85</td>
</tr>
<tr>
<td>fastcgi_index</td>
<td>85</td>
</tr>
<tr>
<td>fastcgi_intercept_errors</td>
<td>86</td>
</tr>
<tr>
<td>fastcgi_keep_conn</td>
<td>86</td>
</tr>
<tr>
<td>fastcgi_max_temp_file_size</td>
<td>86</td>
</tr>
<tr>
<td>fastcgi_next_upstream</td>
<td>86</td>
</tr>
<tr>
<td>fastcgi_no_cache</td>
<td>87</td>
</tr>
<tr>
<td>fastcgi_param</td>
<td>87</td>
</tr>
<tr>
<td>fastcgi_pass</td>
<td>88</td>
</tr>
<tr>
<td>fastcgi_pass_header</td>
<td>88</td>
</tr>
<tr>
<td>fastcgi_read_timeout</td>
<td>89</td>
</tr>
<tr>
<td>fastcgi_pass_request_body</td>
<td>89</td>
</tr>
</tbody>
</table>
mp4_buffer_size ........................................ 130
mp4_max_buffer_size ................................. 130
mp4_limit_rate ....................................... 130
mp4_limit_rate_after ................................. 131

2.29 Module ngx_http_perl_module ................. 132
  2.29.1 Summary ...................................... 132
  2.29.2 Known Bugs .................................. 132
  2.29.3 Example Configuration ...................... 133
  2.29.4 Directives .................................... 134
    perl ............................................... 134
    perl_modules ..................................... 134
    perl_require ..................................... 134
    perl_set ......................................... 135
  2.29.5 Calling Perl from SSI ........................ 135
  2.29.6 The $r Request Object Methods .............. 135

2.30 Module ngx_http_proxy_module ................. 138
  2.30.1 Summary ...................................... 138
  2.30.2 Example Configuration ...................... 138
  2.30.3 Directives .................................... 138
    proxy_bind ...................................... 138
    proxy_buffer_size ................................ 138
    proxy_buffering .................................. 138
    proxy_buffers ................................... 139
    proxy_busy_buffers_size ......................... 139
    proxy_cache ..................................... 139
    proxy_cache_bypass ............................... 140
    proxy_cache_key .................................. 140
    proxy_cache_lock ................................ 140
    proxy_cache_lock_timeout ....................... 140
    proxy_cache_methods .............................. 141
    proxy_cache_min_uses ............................. 141
    proxy_cache_path ................................ 141
    proxy_cache_purge ................................ 142
    proxy_cache_revalidate ......................... 143
    proxy_cache_use_stale ............................ 143
    proxy_cache_valid ............................... 143
    proxy_connect_timeout ......................... 144
    proxy_cookie_domain .............................. 144
    proxy_cookie_path ................................ 145
    proxy_headers_hash_bucket_size ................. 146
    proxy_headers_hash_max_size .................... 146
    proxy_hide_header ................................ 146
    proxy_http_version ............................... 146
    proxy_ignore_client_abort ...................... 146
    proxy_ignore_headers ............................ 147
    proxy_intercept_errors ........................... 147
proxy_max_temp_file_size ........................................ 147
proxy_method .................................................... 148
proxy_next_upstream .............................................. 148
proxy_no_cache .................................................. 149
proxy_pass ........................................................ 149
proxy_pass_header .............................................. 151
proxy_read_timeout .............................................. 151
proxy_pass_request_body ...................................... 151
proxy_pass_request_headers .................................. 151
proxy_redirect .................................................... 152
proxy_send_lowat ............................................... 153
proxy_send_timeout ............................................. 153
proxy_set_body ................................................... 154
proxy_set_header ............................................... 154
proxy_ssl_ciphers .............................................. 154
proxy_ssl_session_reuse ...................................... 155
proxy_ssl_protocols ........................................... 155
proxy_store ........................................................ 155
proxy_store_access ............................................ 156
proxy_temp_file_write_size .................................. 156
proxy_temp_path .................................................. 157
2.30.4 Embedded Variables ..................................... 157

2.31 Module ngx_http_random_index_module .................. 158
  2.31.1 Summary ................................................ 158
  2.31.2 Example Configuration .................................. 158
  2.31.3 Directives ............................................. 158
    random_index ............................................. 158

2.32 Module ngx_http_realip_module ............................ 159
  2.32.1 Summary ................................................ 159
  2.32.2 Example Configuration .................................. 159
  2.32.3 Directives ............................................. 159
    set_real_ip_from ......................................... 159
    real_ip_header ........................................... 159
    real_ip_recursive ........................................ 160

2.33 Module ngx_http_referer_module ........................... 161
  2.33.1 Summary ................................................ 161
  2.33.2 Example Configuration .................................. 161
  2.33.3 Directives ............................................. 161
    referer_hash_bucket_size ................................ 161
    referer_hash_max_size .................................... 161
    valid_referers ............................................. 161

2.34 Module ngx_http_rewrite_module ........................... 163
  2.34.1 Summary ................................................ 163
  2.34.2 Directives ............................................. 163
    break ..................................................... 163
    if ......................................................... 163
2.40 Module ngx_http_ssl_module ........................................ 181
  2.40.1 Summary ......................................................... 181
  2.40.2 Example Configuration ........................................ 181
  2.40.3 Directives ....................................................... 181
    ssl ............................................................... 181
    ssl_buffer_size ................................................. 182
    ssl_certificate .................................................. 182
    ssl_certificate_key .............................................. 182
    ssl_ciphers ..................................................... 183
    ssl_client_certificate ........................................... 183
    ssl_crl .......................................................... 183
    ssl_dhparam ..................................................... 183
    ssl_ecdh_curve .................................................. 184
    ssl_prefer_server_ciphers ...................................... 184
    ssl_protocols ................................................... 184
    ssl_session_cache .............................................. 184
    ssl_session_ticket_key ........................................... 185
    ssl_session_tickets ............................................. 185
    ssl_session_timeout ............................................. 186
    ssl_stapling ..................................................... 186
    ssl_stapling_file ............................................... 186
    ssl_stapling_responder ......................................... 186
    ssl_stapling_verify .............................................. 187
    ssl_trusted_certificate ........................................ 187
    ssl_verify_client ............................................... 187
    ssl_verify_depth ............................................... 187
  2.40.4 Error Processing ............................................... 188
  2.40.5 Embedded Variables ............................................ 188

2.41 Module ngx_http_status_module ................................. 190
  2.41.1 Summary ....................................................... 190
  2.41.2 Example Configuration ........................................ 190
  2.41.3 Directives ....................................................... 190
    status .......................................................... 190
    status_format ................................................... 190
    status_zone ...................................................... 190
  2.41.4 Data ........................................................... 191

2.42 Module ngx_http_sub_module ..................................... 194
  2.42.1 Summary ....................................................... 194
  2.42.2 Example Configuration ........................................ 194
  2.42.3 Directives ....................................................... 194
    sub_filter ....................................................... 194
    sub_filter_last_modified ...................................... 194
    sub_filter_once ................................................. 194
    sub_filter_types ............................................... 195

2.43 Module ngx_http_upstream_module ............................. 196
  2.43.1 Summary ....................................................... 196
2.43.2 Example Configuration ........................................ 196
2.43.3 Directives ..................................................... 196
   upstream ......................................................... 196
   server .......................................................... 197
   zone ............................................................. 198
   ip_hash .......................................................... 199
   keepalive ....................................................... 199
   least_conn ...................................................... 201
   health_check ................................................... 201
   match ............................................................ 203
   queue ............................................................ 204
   sticky ........................................................... 205
   sticky_cookie_insert ........................................... 206
   upstream_conf .................................................. 206
2.43.4 Embedded Variables ........................................ 209

2.44 Module ngx_http_userid_module ................................ 211
   2.44.1 Summary .................................................. 211
   2.44.2 Example Configuration .................................... 211
   2.44.3 Directives .................................................. 211
      userid ......................................................... 211
      userid_domain ................................................. 211
      userid_expires ............................................... 212
      userid_mark ................................................... 212
      userid_name .................................................. 212
      userid_p3p .................................................... 212
      userid_path .................................................. 212
      userid_service ............................................... 213
   2.44.4 Embedded variables ....................................... 213

2.45 Module ngx_http_xslt_module ................................. 214
   2.45.1 Summary .................................................. 214
   2.45.2 Example Configuration .................................... 214
   2.45.3 Directives .................................................. 214
      xml_entities .................................................. 214
      xslt_last_modified ........................................... 214
      xslt_param .................................................... 215
      xslt_string_param ............................................ 215
      xslt_stylesheet ............................................... 215
      xslt_types .................................................... 216

3 Mail server modules .............................................. 217
   3.1 Module ngx_mail_core_module ................................ 217
      3.1.1 Summary .................................................. 217
      3.1.2 Example configuration .................................... 217
      3.1.3 Directives .................................................. 218
         listen ......................................................... 218
         mail .......................................................... 218
3.2 Module ngx_mail_pop3_module .................................. 221
   3.2.1 Directives ............................................. 221
       pop3_auth ............................................. 221
       pop3_capabilities .................................... 221
3.3 Module ngx_mail_imap_module .................................. 222
   3.3.1 Directives ............................................. 222
       imap_auth ............................................. 222
       imap_capabilities .................................... 222
       imap_client_buffer .................................. 222
   3.4 Module ngx_mail_smtp_module .................................. 223
   3.4.1 Directives ............................................. 223
6       smtp_auth ............................................. 223
       smtp_capabilities .................................... 223
3.5 Module ngx_mail_auth_http_module ............................ 224
   3.5.1 Directives ............................................. 224
       auth_http ............................................. 224
       auth_http_header .................................... 224
       auth_http_timeout .................................... 224
   3.5.2 Protocol ............................................... 224
3.6 Module ngx_mail_proxy_module .................................. 227
   3.6.1 Directives ............................................. 227
       proxy_buffer .......................................... 227
       proxy_pass_error_message ............................ 227
       proxy_timeout ........................................ 227
       xclient ................................................ 227
3.7 Module ngx_mail_ssl_module .................................. 229
   3.7.1 Summary ............................................... 229
   3.7.2 Directives ............................................. 229
       ssl ..................................................... 229
       ssl_certificate ....................................... 229
       ssl_certificate_key ................................... 229
       ssl_prefer_server_ciphers ............................ 229
       ssl_protocols ......................................... 230
       ssl_session_cache .................................... 230
       ssl_session_ticket_key ................................ 231
       ssl_session_timeout ................................... 231
       starttls ............................................... 231

A Changelog for NGINX Plus .................................. 232
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B High Availability support</td>
<td>234</td>
</tr>
<tr>
<td>C Legal Notices</td>
<td>235</td>
</tr>
<tr>
<td>Index</td>
<td>238</td>
</tr>
</tbody>
</table>
Chapter 1

Core modules

1.1 Core functionality

1.1.1 Example Configuration

```
user www www;
worker_processes 2;
error_log /var/log/nginx-error.log info;
events {
  use kqueue;
  worker_connections 2048;
}
...
```

1.1.2 Directives

**accept_mutex**

**SYNTAX:** `accept_mutex on | off;`

**DEFAULT:** on

**CONTEXT:** events

If `accept_mutex` is enabled, worker processes will accept new connections by turn. Otherwise, all worker processes will be notified about new connections, and if volume of new connections is low, some of the worker processes may just waste system resources.

The use of `rtsig` connection processing method requires `accept_mutex` to be enabled.

**accept_mutex_delay**

**SYNTAX:** `accept_mutex_delay time;`

**DEFAULT:** 500ms

**CONTEXT:** events
If `accept_mutex` is enabled, specifies the maximum time during which a worker process will try to restart accepting new connections if another worker process is currently accepting new connections.

**daemon**

**SYNTAX:**
```
daemon on | off;
```

**DEFAULT:** `on`

**CONTEXT:** `main`

Determines whether nginx should become a daemon. Mainly used during development.

**debug_connection**

**SYNTAX:**
```
debug_connection address | CIDR | unix::;
```

**DEFAULT:** `—`

**CONTEXT:** `events`

Enables debugging log for selected client connections. Other connections will use logging level set by the `error_log` directive. Debugged connections are specified by IPv4 or IPv6 (1.3.0, 1.2.1) address or network. A connection may also be specified using a hostname. For connections using UNIX-domain sockets (1.3.0, 1.2.1), debugging log is enabled by the “unix:” parameter.

```
events {
    debug_connection 127.0.0.1;
    debug_connection localhost;
    debug_connection 192.0.2.0/24;
    debug_connection ::1;
    debug_connection 2001:0 db8 ::/32;
    debug_connection unix::;
    ...
}
```

For this directive to work, nginx needs to be built with `--with-debug`, see “A debugging log”.

**debug_points**

**SYNTAX:**
```
debug_points abort | stop;
```

**DEFAULT:** `—`

**CONTEXT:** `main`

This directive is used for debugging.

When internal error is detected, e.g. the leak of sockets on restart of working processes, enabling `debug_points` leads to a core file creation (`abort`) or to stopping of a process (`stop`) for further analysis using a system debugger.
error_log

SYNTAX:  error_log file | stderr | syslog:server=address[, parameter=value]
                      [debug | info | notice | warn | error | crit | alert | emerg];

DEFAULT logs/error.log error

CONTEXT: main, http, server, location

Configures logging. Several logs can be specified on the same level (1.5.2). The first parameter defines a file that will store the log.

The special value stderr selects the standard error file. Logging to syslog can be configured by specifying the “syslog:” prefix.

The second parameter determines the level of logging. Log levels above are listed in the order of increasing severity. Setting a certain log level will cause all messages of the specified and more severe log levels to be logged. For example, the default level error will cause error, crit, alert, and emerg messages to be logged. If this parameter is omitted then error is used.

For debug logging to work, nginx needs to be built with --with-debug, see “A debugging log”.

The following parameters configure logging to syslog:

server=address

Defines an address of a syslog server. An address can be specified as a domain name or IP address, and an optional port, or as a UNIX-domain socket path specified after the “unix:” prefix. If port is not specified, the port 514 is used. If a domain name resolves to several IP addresses, the first resolved address is used.

facility=string

Sets facility of syslog messages, as defined in RFC 3164. Facility can be one of “kern”, “user”, “mail”, “daemon”, “auth”, “intern”, “lpr”, “news”, “uucp”, “clock”, “authpriv”, “ftp”, “ntp”, “audit”, “alert”, “cron”, “local0”..“local7”. Default is “local7”.

tag=string

Sets tag of syslog messages. Default is “nginx”.

Example syslog configuration:

```bash
error_log syslog:server=192.168.1.1 debug;
error_log syslog:server=unix:/var/log/nginx.sock;
error_log syslog:server=[2001:db8::1]:12345,facility=local7,tag=nginx error;
```

Logging to syslog is available as part of our commercial subscription.
env

SYNTAX: `env variable[=value];`

DEFAULT `TZ`

CONTEXT: `main`

By default, nginx removes all environment variables inherited from its
parent process except the `TZ` variable. This directive allows preserving some
of the inherited variables, changing their values, or creating new environment
variables. These variables are then:

- inherited during a live upgrade of an executable file;
- used by the `ngx_http_perl_module` module;
- used by worker processes. One should bear in mind that controlling
  system libraries in this way is not always possible as it is common for
  libraries to check variables only during initialization, well before they can
  be set using this directive. An exception from this is an above mentioned
  live upgrade of an executable file.

The `TZ` variable is always inherited and available to the `ngx_http_perl-
module` module, unless it is configured explicitly.

Usage example:

```bash
env MALLOC_OPTIONS;
env PERL5LIB=/data/site/modules;
env OPENSSL_ALLOW_PROXY_CERTS=1;
```

The NGINX environment variable is used internally by nginx and should
not be set directly by the user.

events

SYNTAX: `events { ... }`

DEFAULT —

CONTEXT: `main`

Provides the configuration file context in which the directives that affect
connection processing are specified.

include

SYNTAX: `include file | mask;`

DEFAULT —

CONTEXT: `any`

Includes another `file`, or files matching the specified `mask`, into
configuration. Included files should consist of syntactically correct directives
and blocks.

Usage example:
include mime.types;
include vhosts/*.conf;

lock_file
SYNTAX:  lock_file file;
DEFAULT  logs/nginx.lock
CONTEXT:  main

nginx uses the locking mechanism to implement accept_mutex and serialize access to shared memory. On most systems the locks are implemented using atomic operations, and this directive is ignored. On other systems the “lock file” mechanism is used. This directive specifies a prefix for the names of lock files.

master_process
SYNTAX:  master_process on | off;
DEFAULT  on
CONTEXT:  main

Determines whether worker processes are started. This directive is intended for nginx developers.

multi_accept
SYNTAX:  multi_accept on | off;
DEFAULT  off
CONTEXT:  events

If multi_accept is disabled, a worker process will accept one new connection at a time. Otherwise, a worker process will accept all new connections at a time.

The directive is ignored if kqueue connection processing method is used, because it reports the number of new connections waiting to be accepted.

The use of rtsig connection processing method automatically enables multi_accept.

pcre_jit
SYNTAX:  pcre_jit on | off;
DEFAULT  off
CONTEXT:  main
This directive appeared in version 1.1.12.
Enables or disables the use of “just-in-time compilation” (PCRE JIT) for
the regular expressions known by the time of configuration parsing.

PCRE JIT can speed up processing of regular expressions significantly.

The JIT is available in PCRE libraries starting from version 8.20 built
with the \texttt{--enable-jit} configuration parameter. When the PCRE library
is built with nginx (\texttt{--with-pcre}), the JIT support is enabled via the
\texttt{--with-pcre-jit} configuration parameter.

\textbf{pid}

\begin{verbatim}
SYNTAX:    pid file;
DEFAULT    nginx.pid
CONTEXT:   main
\end{verbatim}

Defines a \textit{file} that will store the process ID of the main process.

\textbf{ssl_engine}

\begin{verbatim}
SYNTAX:    ssl_engine device;
DEFAULT    —
CONTEXT:   main
\end{verbatim}

Defines the name of the hardware SSL accelerator.

\textbf{timer_resolution}

\begin{verbatim}
SYNTAX:    timer_resolution interval;
DEFAULT    —
CONTEXT:   main
\end{verbatim}

Reduces timer resolution in worker processes, thus reducing the number
of \texttt{gettimeofday} system calls made. By default, \texttt{gettimeofday} is called each
time a kernel event is received. With reduced resolution, \texttt{gettimeofday} is only
called once per specified \textit{interval}.

Example:

\begin{verbatim}
timer_resolution 100ms;
\end{verbatim}

Internal implementation of the interval depends on the method used:

- the \texttt{EVFILT\_TIMER} filter if \texttt{kqueue} is used;
- \texttt{timer\_create} if \texttt{eventport} is used;
- \texttt{setitimer} otherwise.
**use**

**SYNTAX:** use method;
**DEFAULT:** —
**CONTEXT:** events

Specifies the connection processing method to use. There is normally no need to specify it explicitly, because nginx will by default use the most efficient method.

**user**

**SYNTAX:** user user [group];
**DEFAULT:** nobody nobody
**CONTEXT:** main

Defines user and group credentials used by worker processes. If group is omitted, a group whose name equals that of user is used.

**worker_aio_requests**

**SYNTAX:** worker_aio_requests number;
**DEFAULT:** 32
**CONTEXT:** events

This directive appeared in versions 1.1.4 and 1.0.7.

When using aio with the epoll connection processing method, sets the maximum number of outstanding asynchronous I/O operations for a single worker process.

**worker_connections**

**SYNTAX:** worker_connections number;
**DEFAULT:** 512
**CONTEXT:** events

Sets the maximum number of simultaneous connections that can be opened by a worker process.

It should be kept in mind that this number includes all connections (e.g. connections with proxied servers, among others), not only connections with clients. Another consideration is that the actual number of simultaneous connections cannot exceed the current limit on the maximum number of open files, which can be changed by worker_rlimit_nofile.

**worker_cpu_affinity**

**SYNTAX:** worker_cpu_affinity cpumask ...;
**DEFAULT:** —
**CONTEXT:** main
Binds worker processes to the sets of CPUs. Each CPU set is represented by a bitmask of allowed CPUs. There should be a separate set defined for each of the worker processes. By default, worker processes are not bound to any specific CPUs.

For example,

```
worker_processes 4;
worker_cpu_affinity 0001 0010 0100 1000;
```

binds each worker process to a separate CPU, while

```
worker_processes 2;
worker_cpu_affinity 0101 1010;
```

binds the first worker process to CPU0/CPU2, and the second worker process to CPU1/CPU3. The second example is suitable for hyper-threading.

The directive is only available on FreeBSD and Linux.

**worker_priority**

**SYNTAX:** \texttt{worker\_priority number};

**DEFAULT:** 0

**CONTEXT:** main

Defines the scheduling priority for worker processes like it is done by the \texttt{nice} command: a negative \texttt{number} means higher priority. Allowed range normally varies from -20 to 20.

Example:

```
worker_priority -10;
```

**worker_processes**

**SYNTAX:** \texttt{worker\_processes number | auto};

**DEFAULT:** 1

**CONTEXT:** main

Defines the number of worker processes.

The optimal value depends on many factors including (but not limited to) the number of CPU cores, the number of hard disk drives that store data, and load pattern. When one is in doubt, setting it to the number of available CPU cores would be a good start (the value \texttt{auto} will try to autodetect it).

The \texttt{auto} parameter is supported starting from versions 1.3.8 and 1.2.5.
worker_rlimit_core

SYNTAX:  worker_rlimit_core size;
DEFAULT  —
CONTEXT: main

Changes the limit on the largest size of a core file (RLIMIT_CORE) for worker processes. Used to increase the limit without restarting the main process.

worker_rlimit_nofile

SYNTAX:  worker_rlimit_nofile number;
DEFAULT  —
CONTEXT: main

Changes the limit on the maximum number of open files (RLIMIT_NOFILE) for worker processes. Used to increase the limit without restarting the main process.

worker_rlimit_sigpending

SYNTAX:  worker_rlimit_sigpending number;
DEFAULT  —
CONTEXT: main

On systems that support rtsig connection processing method, changes the limit on the number of signals that may be queued (RLIMIT_SIGPENDING) for worker processes. Used to increase the limit without restarting the main process.

working_directory

SYNTAX:  working_directory directory;
DEFAULT  —
CONTEXT: main

Defines the current working directory for a worker process. It is primarily used when writing a core-file, in which case a worker process should have write permission for the specified directory.
1.2 Setting up hashes

1.2.1 Overview

To quickly process static sets of data such as server names, map directive’s values, MIME types, names of request header strings, nginx uses hash tables. During the start and each re-configuration nginx selects the minimum possible sizes of hash tables such that the bucket size that stores keys with identical hash values does not exceed the configured parameter (hash bucket size). The size of a table is expressed in buckets. The adjustment is continued until the table size exceeds the hash max size parameter. Most hashes have the corresponding directives that allow to change these parameters, for example, for the server names hash they are server_names_hash_max_size and server_names_hash_bucket_size.

The hash bucket size parameter is aligned to the size that is a multiple of the processor’s cache line size. This speeds up key search in a hash on modern processors by reducing the number of memory accesses. If hash bucket size is equal to one processor’s cache line size then the number of memory accesses during the key search will be two in the worst case — first to compute the bucket address, and second during the key search inside the bucket. Therefore, if nginx emits the message requesting to increase either hash max size or hash bucket size then the first parameter should first be increased.
1.3 Connection processing methods

1.3.1 Overview

nginx supports a variety of connection processing methods. The availability of a particular method depends on the platform used. On platforms that support several methods nginx will normally select the most efficient method automatically. However, if needed, a connection processing method can be selected explicitly with the use directive.

The following connection processing methods are supported:

- **select** — standard method. The supporting module is built automatically on platforms that lack more efficient methods. The --with-select_module and --without-select_module configuration parameters can be used to forcibly enable or disable the build of this module.

- **poll** — standard method. The supporting module is built automatically on platforms that lack more efficient methods. The --with-poll_module and --without-poll_module configuration parameters can be used to forcibly enable or disable the build of this module.

- **kqueue** — efficient method used on FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0, and Mac OS X.

- **epoll** — efficient method used on Linux 2.6+.

Some older distributions like SuSE 8.2 provide patches that add epoll support to 2.4 kernels.

- **rtsig** — real time signals, efficient method used on Linux 2.2.19+. By default, the system-wide event queue is limited by 1024 signals. On loaded servers it may become necessary to increase this limit by changing the /proc/sys/kernel/rtsig-max kernel parameter. However, in Linux 2.6.6-mm2 this parameter is gone, and each process now has its own event queue. The size of each queue is limited by RLIMIT_SIGPENDING and can be changed with worker_rlimit_sigpending.

On queue overflow, nginx discards the queue and falls back to poll connection processing method until the situation gets back to normal.

- **/dev/poll** — efficient method used on Solaris 7 11/99+, HP/UX 11.22+ (eventport), IRIX 6.5.15+, and Tru64 UNIX 5.1A+.

- **eventport** — event ports, efficient method used on Solaris 10.
Chapter 2

HTTP server modules

2.1 Module ngx_http_core_module

2.1.1 Directives

aio

SYNTAX: aio on | off | sendfile;
DEFAULT off
CONTEXT: http, server, location
This directive appeared in version 0.8.11.

Enables or disables the use of asynchronous file I/O (AIO) on FreeBSD and Linux.

On FreeBSD, AIO can be used starting from FreeBSD 4.3. AIO can either be linked statically into a kernel:

```
options VFS_AIO
```

or loaded dynamically as a kernel loadable module:

```
kldload aio
```

In FreeBSD versions 5 and 6, enabling AIO statically, or dynamically when booting the kernel, will cause the entire networking subsystem to use the Giant lock, which can impact overall performance negatively. This limitation has been removed in FreeBSD 6.4-STABLE in 2009, and in FreeBSD 7. However, starting from FreeBSD 5.3 it is possible to enable AIO without the penalty of running the networking subsystem under a Giant lock - for this to work, the AIO module needs to be loaded after the kernel has booted. In this case, the following message will appear in /var/log/messages

```
WARNING: Network stack Giant-free, but aio requires Giant.
Consider adding 'options NET_WITH_GIANT' or setting debug.mpsafenet=0
```

and can safely be ignored.
The requirement to use the Giant lock with AIO is related to the fact
that FreeBSD supports asynchronous calls `aio_read` and `aio_write` when
working with sockets. However, since nginx uses AIO only for disk I/O, no
problems should arise.

For AIO to work, `sendfile` needs to be disabled:

```language-http
location /video/ {
    sendfile  off;
    aio       on;
    output_buffers 1 64k;
}
```

In addition, starting from FreeBSD 5.2.1 and nginx 0.8.12, AIO can also
be used to pre-load data for `sendfile`:

```language-http
location /video/ {
    sendfile  on;
    tcp_nopush on;
    aio       sendfile;
}
```

In this configuration, `sendfile` is called with the `SF_NODISKIO` flag which
causes it not to block on disk I/O, but, instead, report back that the data are
not in memory. nginx then initiates an asynchronous data load by reading one
byte. On the first read, the FreeBSD kernel loads the first 128K bytes of a file
into memory, although next reads will only load data in 16K chunks. This can
be changed using the `read_ahead` directive.

On Linux, AIO can be used starting from kernel version 2.6.22. Also, it is
necessary to enable `directio`, or otherwise reading will be blocking:

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

On Linux, `directio` can only be used for reading blocks that are aligned on
512-byte boundaries (or 4K for XFS). File’s unaligned end is read in blocking
mode. The same holds true for byte range requests and for FLV requests not
from the beginning of a file: reading of unaligned data at the beginning and
end of a file will be blocking. There is no need to turn off `sendfile` explicitly,
as it is turned off automatically when `directio` is used.

**alias**

**SYNTAX:** `alias path;`

**DEFAULT** —

**CONTEXT:** `location`

Defines a replacement for the specified location. For example, with the
following configuration
on request of “/i/top.gif”, the file /data/w3/images/top.gif will be sent.

The path value can contain variables, except $document_root and $realpath_root.

If alias is used inside a location defined with a regular expression then such regular expression should contain captures and alias should refer to these captures (0.7.40), for example:

```
location ~ ^/users/(.*)\.(?:gif|jpe?g|png)$ {  
  alias /data/w3/images/$1;  
}
```

When location matches the last part of the directive’s value:

```
location /images/ {  
  alias /data/w3/images/;  
}
```

it is better to use the root directive instead:

```
location /images/ {  
  root /data/w3;  
}
```

**chunked Transfer Encoding**

**SYNTAX:** chunked_transfer_encoding on | off;

**DEFAULT** on

**CONTEXT:** http, server, location

Allows disabling chunked transfer encoding in HTTP/1.1. It may come in handy when using a software failing to support chunked encoding despite the standard’s requirement.

**Client Body Buffer Size**

**SYNTAX:** client_body_buffer_size size;

**DEFAULT** 8k|16k

**CONTEXT:** http, server, location

Sets buffer size for reading client request body. In case the request body is larger than the buffer, the whole body or only its part is written to a temporary file. By default, buffer size is equal to two memory pages. This is 8K on x86, other 32-bit platforms, and x86-64. It is usually 16K on other 64-bit platforms.
client_body_in_file_only

SYNTAX:  client_body_in_file_only on | clean | off;
DEFAULT  off
CONTEXT:  http, server, location

Determines whether nginx should save the entire client request body into a file. This directive can be used during debugging, or when using the $request_body_file variable, or the $r->request_body_file method of the module ngx_http_perl_module.

When set to the value on, temporary files are not removed after request processing.

The value clean will cause the temporary files left after request processing to be removed.

client_body_in_single_buffer

SYNTAX:  client_body_in_single_buffer on | off;
DEFAULT  off
CONTEXT:  http, server, location

Determines whether nginx should save the entire client request body in a single buffer. The directive is recommended when using the $request_body variable, to save the number of copy operations involved.

client_body_temp_path

SYNTAX:  client_body_temp_path path [level1 [level2 |level3]];
DEFAULT  client_body_temp
CONTEXT:  http, server, location

Defines a directory for storing temporary files holding client request bodies. Up to three-level subdirectory hierarchy can be used under the specified directory. For example, in the following configuration

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

a path to a temporary file might look like this:

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

client_body_timeout

SYNTAX:  client_body_timeout time;
DEFAULT  60s
CONTEXT:  http, server, location

Defines a timeout for reading client request body. The timeout is set only for a period between two successive read operations, not for the transmission
of the whole request body. If a client does not transmit anything within this time, the 408 Request Time-out error is returned to the client.

**client_header_buffer_size**

- **SYNTAX:** `client_header_buffer_size size;`
- **DEFAULT:** 1k
- **CONTEXT:** http, server

Sets buffer size for reading client request header. For most requests, a buffer of 1K bytes is enough. However, if a request includes long cookies, or comes from a WAP client, it may not fit into 1K. If a request line or a request header field does not fit into this buffer then larger buffers, configured by the `large_client_header_buffers` directive, are allocated.

**client_header_timeout**

- **SYNTAX:** `client_header_timeout time;`
- **DEFAULT:** 60s
- **CONTEXT:** http, server

Defines a timeout for reading client request header. If a client does not transmit the entire header within this time, the 408 Request Time-out error is returned to the client.

**client_max_body_size**

- **SYNTAX:** `client_max_body_size size;`
- **DEFAULT:** 1m
- **CONTEXT:** http, server, location

Sets the maximum allowed size of the client request body, specified in the `Content-Length` request header field. If the size in a request exceeds the configured value, the 413 Request Entity Too Large error is returned to the client. Please be aware that browsers cannot correctly display this error. Setting `size` to 0 disables checking of client request body size.

**connection_pool_size**

- **SYNTAX:** `connection_pool_size size;`
- **DEFAULT:** 256
- **CONTEXT:** http, server

Allows accurate tuning of per-connection memory allocations. This directive has minimal impact on performance and should not generally be used.
CHAPTER 2. HTTP SERVER MODULES 2.1. MODULE NGX_HTTP_CORE_MODULE

**default_type**

SYNTAX:  
\[ \text{default_type } \text{mime-type}; \]

DEFAULT:  
text/plain

CONTEXT: http, server, location

Defines the default MIME type of a response. Mapping of file name extensions to MIME types can be set with the `types` directive.

**directio**

SYNTAX:  
\[ \text{directio } \text{size} \mid \text{off}; \]

DEFAULT:  
off

CONTEXT: http, server, location

This directive appeared in version 0.7.7.

Enables the use of the \texttt{O\_DIRECT} flag (FreeBSD, Linux), the \texttt{F\_NOCACHE} flag (Mac OS X), or the \texttt{directio} function (Solaris), when reading files that are larger than or equal to the specified \textit{size}. The directive automatically disables (0.7.15) the use of \texttt{sendfile} for a given request. It can be useful for serving large files:

\begin{verbatim}
directio 4m;
\end{verbatim}

or when using \texttt{aio} on Linux.

**directio_alignment**

SYNTAX:  
\[ \text{directio_alignment } \text{size}; \]

DEFAULT:  
512

CONTEXT: http, server, location

This directive appeared in version 0.8.11.

Sets the alignment for \texttt{directio}. In most cases, a 512-byte alignment is enough. However, when using XFS under Linux, it needs to be increased to 4K.

**disable_symlinks**

SYNTAX:  
\[ \text{disable_symlinks } \text{off}; \]

SYNTAX:  
\[ \text{disable_symlinks } \text{on} \mid \text{if_not_owner [from=part]}; \]

DEFAULT:  
off

CONTEXT: http, server, location

This directive appeared in version 1.1.15.

Determines how symbolic links should be treated when opening files:

\textit{off}

Symbolic links in the pathname are allowed and not checked. This is the default behavior.
If any component of the pathname is a symbolic link, access to a file is denied.

**if_not_owner**
Access to a file is denied if any component of the pathname is a symbolic link, and the link and object that the link points to have different owners.

**from=part**
When checking symbolic links (parameters on and if_not_owner), all components of the pathname are normally checked. Checking of symbolic links in the initial part of the pathname may be avoided by specifying additionally the from=part parameter. In this case, symbolic links are checked only from the pathname component that follows the specified initial part. If the value is not an initial part of the pathname checked, the whole pathname is checked as if this parameter was not specified at all. If the value matches the whole file name, symbolic links are not checked. The parameter value can contain variables.

Example:

```
disable_symlinks on from=$document_root;
```

This directive is only available on systems that have the openat and fstatat interfaces. Such systems include modern versions of FreeBSD, Linux, and Solaris.

Parameters on and if_not_owner add a processing overhead.

On systems that do not support opening of directories only for search, to use these parameters it is required that worker processes have read permissions for all directories being checked.

The ngx_http_autoindex_module, ngx_http_random_index_module, and ngx_http_dav_module modules currently ignore this directive.

**error_page**

**SYNTAX:** error_page code ...[=response] uri;

**DEFAULT**

**CONTEXT:** http, server, location, if in location

Defines the URI that will be shown for the specified errors. error_page directives are inherited from the previous level only if there are no error_page directives defined on the current level. A uri value can contain variables.

Example:

```
error_page 404 /404.html;
error_page 500 502 503 504 /50x.html;
```
Furthermore, it is possible to change the response code to another using the "=response" syntax, for example:

```plaintext
error_page 404 =200 /empty.gif;
```

If an error response is processed by a proxied server or a FastCGI server, and the server may return different response codes (e.g., 200, 302, 401 or 404), it is possible to respond with the code it returns:

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

It is also possible to use redirects for error processing:

```plaintext
error_page 403 http://example.com/forbidden.html;
error_page 404 =301 http://example.com/notfound.html;
```

In this case, by default, the response code 302 is returned to the client. It can only be changed to one of the redirect status codes (301, 302, 303, and 307).

If there is no need to change URI during internal redirection it is possible to pass error processing into a named location:

```plaintext
location / {
    error_page 404 = @fallback;
}

location @fallback {
    proxy_pass http://backend;
}
```

If uri processing leads to an error, the status code of the last occurred error is returned to the client.

**etag**

**SYNTAX:**  
etag on | off;

**DEFAULT**  
on

**CONTEXT:**  
http, server, location

This directive appeared in version 1.3.3.

Enables or disables automatic generation of the ETag response header field for static resources.

**http**

**SYNTAX:**  
http { ...}

**DEFAULT**  
—

**CONTEXT:**  
main

Provides the configuration file context in which the HTTP server directives are specified.
if_modified_since

SYNTAX: if_modified_since off | exact | before;
DEFAULT exact
CONTEXT: http, server, location

This directive appeared in version 0.7.24.

Specifies how to compare modification time of a response with the time in the If-Modified-Since request header field:

off
  the If-Modified-Since request header field is ignored (0.7.34);
exact
  exact match;
before
  modification time of a response is less than or equal to the time in the If-Modified-Since request header field.

ignore_invalid_headers

SYNTAX: ignore_invalid_headers on | off;
DEFAULT on
CONTEXT: http, server

Controls whether header fields with invalid names should be ignored. Valid names are composed of English letters, digits, hyphens, and possibly underscores (as controlled by the underscores_in_headers directive).

If the directive is specified on the server level, its value is only used if a server is a default one. The value specified also applies to all virtual servers listening on the same address and port.

internal

SYNTAX: internal;
DEFAULT —
CONTEXT: location

Specifies that a given location can only be used for internal requests. For external requests, the client error 404 Not Found is returned. Internal requests are the following:

- requests redirected by the error_page, index, random_index, and try_files directives;
- requests redirected by the X-Accel-Redirect response header field from an upstream server;
- subrequests formed by the “include virtual” command of the ngx_http_ssi_module module and by the ngx_http_addition_module module directives;
• requests changed by the rewrite directive.

Example:

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

There is a limit of 10 internal redirects per request to prevent request processing cycles that can occur in incorrect configurations. If this limit is reached, the error 500 Internal Server Error is returned. In such cases, the “rewrite or internal redirection cycle” message can be seen in the error log.

keepalive_disable

SYNTAX:  keepalive_disable none | browser ...;
DEFAULT  msie6
CONTEXT: http, server, location

Disables keep-alive connections with misbehaving browsers. The browser parameters specify which browsers will be affected. The value msie6 disables keep-alive connections with old versions of MSIE, once a POST request is received. The value safari disables keep-alive connections with Safari and Safari-like browsers on Mac OS X and Mac OS X-like operating systems. The value none enables keep-alive connections with all browsers.

Prior to version 1.1.18, the value safari matched all Safari and Safari-like browsers on all operating systems, and keep-alive connections with them were disabled by default.

keepalive_requests

SYNTAX:  keepalive_requests number;
DEFAULT  100
CONTEXT: http, server, location

This directive appeared in version 0.8.0.

Sets the maximum number of requests that can be served through one keep-alive connection. After the maximum number of requests are made, the connection is closed.
**keepalive_timeout**

**SYNTAX:**  
keepalive_timeout timeout [header_timeout];

**DEFAULT:** 75s

**CONTEXT:** http, server, location

The first parameter sets a timeout during which a keep-alive client connection will stay open on the server side. The zero value disables keep-alive client connections. The optional second parameter sets a value in the `Keep-Alive: timeout=time` response header field. Two parameters may differ.

The `Keep-Alive: timeout=time` header field is recognized by Mozilla and Konqueror. MSIE closes keep-alive connections by itself in about 60 seconds.

**large_client_header_buffers**

**SYNTAX:**  
large_client_header_buffers number size;

**DEFAULT:** 4 8k

**CONTEXT:** http, server

Sets the maximum `number` and `size` of buffers used for reading large client request header. A request line cannot exceed the size of one buffer, or the 414 Request-URI Too Large error is returned to the client. A request header field cannot exceed the size of one buffer as well, or the 400 Bad Request error is returned to the client. Buffers are allocated only on demand. By default, the buffer size is equal to 8K bytes. If after the end of request processing a connection is transitioned into the keep-alive state, these buffers are released.

**limit_except**

**SYNTAX:**  
limit_except method ...{...}

**DEFAULT:** —

**CONTEXT:** location

Limits allowed HTTP methods inside a location. The `method` parameter can be one of the following: GET, HEAD, POST, PUT, DELETE, MKCOL, COPY, MOVE, OPTIONS, PROPFIND, PROPPATCH, LOCK, UNLOCK, or PATCH. Allowing the GET method makes the HEAD method also allowed. Access to other methods can be limited using the `ngx_http_access_module` and `ngx_http_auth_basic_module` modules directives:

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

Please note that this will limit access to all methods except GET and HEAD.
**limit_rate**

**SYNTAX:** `limit_rate rate;`

**DEFAULT:** 0

**CONTEXT:** http, server, location, if in location

Limits the rate of response transmission to a client. The `rate` is specified in bytes per second. The zero value disables rate limiting.

The limit is set per a request, and so if a client simultaneously opens two connections, the overall rate will be twice as much as the specified limit.

Rate limit can also be set in the `$limit_rate` variable. It may be useful in cases where rate should be limited depending on a certain condition:

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

Rate limit can also be set in the `X-Accel-Limit-Rate` header field of a proxied server response. This capability can be disabled using the `proxy_ignore_headers` and `fastcgi_ignore_headers` directives.

**limit_rate_after**

**SYNTAX:** `limit_rate_after size;`

**DEFAULT:** 0

**CONTEXT:** http, server, location, if in location

This directive appeared in version 0.8.0.

Sets the initial amount after which the further transmission of a response to a client will be rate limited.

Example:

```plaintext
location /flv/ {
    flv;
    limit_rate_after 500k;
    limit_rate 50k;
}
```

**lingering_close**

**SYNTAX:** `lingering_close off | on | always;`

**DEFAULT:** on

**CONTEXT:** http, server, location

This directive appeared in versions 1.1.0 and 1.0.6.

Controls how `nginx` closes client connections.
The default value “on” instructs nginx to wait for and process additional data from a client before fully closing a connection, but only if heuristics suggests that a client may be sending more data.

The value “always” will cause nginx to unconditionally wait for and process additional client data.

The value “off” tells nginx to never wait for more data and close the connection immediately. This behavior breaks the protocol and should not be used under normal circumstances.

**lingering_time**

**Syntax:**

```
lingering_time time;
```

**Default:** 30s

**Context:** http, server, location

When `lingering_close` is in effect, this directive specifies the maximum time during which nginx will process (read and ignore) additional data coming from a client. After that, the connection will be closed, even if there will be more data.

**lingering_timeout**

**Syntax:**

```
lingering_timeout time;
```

**Default:** 5s

**Context:** http, server, location

When `lingering_close` is in effect, this directive specifies the maximum waiting time for more client data to arrive. If data are not received during this time, the connection is closed. Otherwise, the data are read and ignored, and nginx starts waiting for more data again. The “wait-read-ignore” cycle is repeated, but no longer than specified by the `lingering_time` directive.

**listen**

**Syntax:**

```
listen address[:port] [default_server] [ssl] [spdy] [proxy_protocol]
[setfib=number] [fastopen=number] [backlog=number] [rcvbuf=size]
[sndbuf=size] [accept_filter=filter] [deferred] [bind]
[ipv6only=on|off]
[so_keepalive=on|off] [keepidle]:[keepintvl]:[keepcnt];
```

**Syntax:**

```
listen port [default_server] [ssl] [spdy] [proxy_protocol]
[setfib=number] [fastopen=number] [backlog=number] [rcvbuf=size]
[sndbuf=size] [accept_filter=filter] [deferred] [bind]
[ipv6only=on|off]
[so_keepalive=on|off] [keepidle]:[keepintvl]:[keepcnt];
```

**Syntax:**

```
listen unix: path [default_server] [ssl] [spdy] [proxy_protocol]
[backlog=number] [rcvbuf=size] [sndbuf=size] [accept_filter=filter]
[deferred] [bind] [so_keepalive=on|off] [keepidle]:[keepintvl]:[keepcnt];
```

**Default:** *

**Context:** server
Sets the address and port for IP, or the path for a UNIX-domain socket on which the server will accept requests. Both address and port, or only address or only port can be specified. An address may also be a hostname, for example:

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

IPv6 addresses (0.7.36) are specified in square brackets:

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

UNIX-domain sockets (0.8.21) are specified with the “unix:” prefix:

```
listen unix:/var/run/nginx.sock;
```

If only address is given, the port 80 is used.
If the directive is not present then either *:80 is used if nginx runs with the superuser privileges, or *:8000 otherwise.

The default_server parameter, if present, will cause the server to become the default server for the specified address:port pair. If none of the directives have the default_server parameter then the first server with the address:port pair will be the default server for this pair.

In versions prior to 0.8.21 this parameter is named simply default.

The ssl parameter (0.7.14) allows specifying that all connections accepted on this port should work in SSL mode. This allows for a more compact configuration for the server that handles both HTTP and HTTPS requests.

The spdy parameter (1.3.15) allows accepting SPDY connections on this port. Normally, for this to work the ssl parameter should be specified as well, but nginx can also be configured to accept SPDY connections without SSL.

The proxy_protocol parameter (1.5.12) allows specifying that all connections accepted on this port should use the PROXY protocol.

A listen directive can have several additional parameters specific to socket-related system calls. These parameters can be specified in any listen directive, but only once for a given address:port pair.

In versions prior to 0.8.21, they could only be specified in the listen directive together with the default parameter.

```
setfib=number
```

this parameter (0.8.44) sets the associated routing table, FIB (the SO_SETFIB option) for the listening socket. This currently works only on FreeBSD.
fastopen=number
enables “TCP Fast Open” for the listening socket (1.5.8) and limits the maximum length for the queue of connections that have not yet completed the three-way handshake.

Do not enable this feature unless the server can handle receiving the same SYN packet with data more than once.

backlog=number
sets the backlog parameter in the listen call that limits the maximum length for the queue of pending connections. By default, backlog is set to -1 on FreeBSD and Mac OS X, and to 511 on other platforms.

cvbuf=size
sets the receive buffer size (the SO_RCVBUF option) for the listening socket.

sndbuf=size
sets the send buffer size (the SO_SNDBUF option) for the listening socket.

accept_filter=filter
sets the name of accept filter (the SO_ACCEPTFILTER option) for the listening socket that filters incoming connections before passing them to accept. This works only on FreeBSD and NetBSD 5.0+. Possible values are dataready and httpready.

defered
instructs to use a deferred accept (the TCP_DEFER_ACCEPT socket option) on Linux.

bind
instructs to make a separate bind call for a given address:port pair. This is useful because if there are several listen directives with the same port but different addresses, and one of the listen directives listens on all addresses for the given port (*:port), nginx will bind only to *:port. It should be noted that the getsockname system call will be made in this case to determine the address that accepted the connection. If the setfib, backlog, cvbuf, sndbuf, accept_filter, deferred, or so_keepalive parameters are used then for a given address:port pair a separate bind call will always be made.

ipv6only=on|off
this parameter (0.7.42) determines (via the IPV6_V6ONLY socket option) whether an IPv6 socket listening on a wildcard address [:] will accept only IPv6 connections or both IPv6 and IPv4 connections. This parameter is turned on by default. It can only be set once on start.

Prior to version 1.3.4, if this parameter was omitted then the operating system’s settings were in effect for the socket.

so_keepalive=on|off|[keepidle]:[keepntvl]:[keepcnt]
this parameter (1.1.11) configures the “TCP keepalive” behavior for the listening socket. If this parameter is omitted then the operating system’s
settings will be in effect for the socket. If it is set to the value “on”, the SO_KEEPALIVE option is turned on for the socket. If it is set to the value “off”, the SO_KEEPALIVE option is turned off for the socket. Some operating systems support setting of TCP keepalive parameters on a per-socket basis using the TCP_KEEPIDLE, TCP_KEEPINTVL, and TCP_KEEPCNT socket options. On such systems (currently, Linux 2.4+, NetBSD 5+, and FreeBSD 9.0-STABLE), they can be configured using the keepidle, keepintvl, and keepcnt parameters. One or two parameters may be omitted, in which case the system default setting for the corresponding socket option will be in effect. For example,

so_keepalive=30m::10

will set the idle timeout (TCP_KEEPIDLE) to 30 minutes, leave the probe interval (TCP_KEEPINTVL) at its system default, and set the probes count (TCP_KEEPCNT) to 10 probes.

Example:

listen 127.0.0.1 default_server accept_filter=dataready backlog=1024;

location

SYNTAX:  location [ = | ~ | ~* | ~* ] uri { ... }
SYNTAX:  location @name { ... }
DEFAULT —
CONTEXT:  server, location

Sets configuration depending on a request URI.

The matching is performed against a normalized URI, after decoding the text encoded in the “%XX” form, resolving references to relative path components “.” and “..”, and possible compression of two or more adjacent slashes into a single slash.

A location can either be defined by a prefix string, or by a regular expression. Regular expressions are specified with the preceding “~*” modifier (for case-insensitive matching), or the “~” modifier (for case-sensitive matching). To find location matching a given request, nginx first checks locations defined using the prefix strings (prefix locations). Among them, the location with the longest matching prefix is selected and remembered. Then regular expressions are checked, in the order of their appearance in the configuration file. The search of regular expressions terminates on the first match, and the corresponding configuration is used. If no match with a regular expression is found then the configuration of the prefix location remembered earlier is used.

location blocks can be nested, with some exceptions mentioned below.

For case-insensitive operating systems such as Mac OS X and Cygwin, matching with prefix strings ignores a case (0.7.7). However, comparison is limited to one-byte locales.
Regular expressions can contain captures (0.7.40) that can later be used in other directives.

If the longest matching prefix location has the “^~” modifier then regular expressions are not checked.

Also, using the “=” modifier it is possible to define an exact match of URI and location. If an exact match is found, the search terminates. For example, if a “/” request happens frequently, defining “location = /” will speed up the processing of these requests, as search terminates right after the first comparison. Such a location cannot obviously contain nested locations.

In versions from 0.7.1 to 0.8.41, if a request matched the prefix location without the “=” and “^~” modifiers, the search also terminated and regular expressions were not checked.

Let’s illustrate the above by an example:

```
location = / {
  [ configuration A ]
}

location / {
  [ configuration B ]
}

location /documents/ {
  [ configuration C ]
}

location ^~ /images/ {
  [ configuration D ]
}

location ~* \.(gif|jpg|jpeg)$ {
  [ configuration E ]
}
```

The “/” request will match configuration A, the “/index.html” request will match configuration B, the “/documents/document.html” request will match configuration C, the “/images/1.gif” request will match configuration D, and the “/documents/1.jpg” request will match configuration E.

The “@” prefix defines a named location. Such a location is not used for a regular request processing, but instead used for request redirection. They cannot be nested, and cannot contain nested locations.

If a location is defined by a prefix string that ends with the slash character, and requests are processed by one of proxy_pass, fastcgi_pass, scgi_pass, uwsgi_pass, or memcached_pass, then in response to a request with URI equal to this string, but without the trailing slash, a permanent redirect with the code 301 will be returned to the requested URI with the slash appended. If this is not desired, an exact match of the URI and location could be defined like this:

```
location /user/ {
  proxy_pass http://user.example.com;
}
```
location = /user {
    proxy_pass http://login.example.com;
}

log_not_found

SYNTAX:  log_not_found on | off;
DEFAULT  on
CONTEXT: http, server, location

Enables or disables logging of errors about not found files into error_log.

log_subrequest

SYNTAX:  log_subrequest on | off;
DEFAULT  off
CONTEXT: http, server, location

Enables or disables logging of subrequests into access_log.

max_ranges

SYNTAX:  max_ranges number;
DEFAULT  —
CONTEXT: http, server, location

This directive appeared in version 1.1.2.

Limits the maximum allowed number of ranges in byte-range requests. Requests that exceed the limit are processed as if there were no byte ranges specified. By default, the number of ranges is not limited. The zero value disables the byte-range support completely.

merge_slashes

SYNTAX:  merge_slashes on | off;
DEFAULT  on
CONTEXT: http, server

Enables or disables compression of two or more adjacent slashes in a URI into a single slash.

Note that compression is essential for the correct matching of prefix string and regular expression locations. Without it, the “/scripts/one.php” request would not match

location /scripts/ {
    ...
}

and might be processed as a static file. So it gets converted to “/scripts/one.php”.

Nginx, Inc.  p.46 of 242
Turning the compression off can become necessary if a URI contains base64-encoded names, since base64 uses the “/” character internally. However, for security considerations, it is better to avoid turning the compression off.

If the directive is specified on the server level, its value is only used if a server is a default one. The value specified also applies to all virtual servers listening on the same address and port.

**msie_padding**

SYNTAX: `msie_padding on | off;`

DEFAULT `on`

CONTEXT: `http, server, location`

Enables or disables adding comments to responses for MSIE clients with status greater than 400 to increase the response size to 512 bytes.

**msie_refresh**

SYNTAX: `msie_refresh on | off;`

DEFAULT `off`

CONTEXT: `http, server, location`

Enables or disables issuing refreshes instead of redirects for MSIE clients.

**open_file_cache**

SYNTAX: `open_file_cache off;`

SYNTAX: `open_file_cache max=N [inactive=time];`

DEFAULT `off`

CONTEXT: `http, server, location`

Configures a cache that can store:

- open file descriptors, their sizes and modification times;
- information on existence of directories;
- file lookup errors, such as “file not found”, “no read permission”, and so on.

Caching of errors should be enabled separately by the `open_file_cache_errors` directive.

The directive has the following parameters:

**max**

sets the maximum number of elements in the cache; on cache overflow the least recently used (LRU) elements are removed;

**inactive**

defines a time after which an element is removed from the cache if it has not been accessed during this time; by default, it is 60 seconds;
off
disables the cache.

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`

Enables or disables caching of file lookup errors by `open_file_cache`.

**open_file_cache_min_uses**

SYNTAX: `open_file_cache_min_uses number;`

DEFAULT: `1`

CONTEXT: `http, server, location`

Sets the minimum `number` of file accesses during the period configured by the `inactive` parameter of the `open_file_cache` directive, required for a file descriptor to remain open in the cache.

**open_file_cache_valid**

SYNTAX: `open_file_cache_valid time;`

DEFAULT: `60s`

CONTEXT: `http, server, location`

Sets a time after which `open_file_cache` elements should be validated.

**optimize_server_names**

SYNTAX: `optimize_server_names on | off;`

DEFAULT: `off`

CONTEXT: `http, server`

This directive is obsolete. The `server_name_in_redirect` directive should be used instead.

**output_buffers**

SYNTAX: `output_buffers number size;`

DEFAULT: `1 32k`

CONTEXT: `http, server, location`

Sets the `number` and `size` of buffers used for reading a response from a disk.
port_in_redirect
SYNTAX:  port_in_redirect on | off;
DEFAULT  on
CONTEXT: http, server, location

Enables or disables specifying the port in redirects issued by nginx.
The use of the primary server name in redirects is controlled by the server_name_in_redirect directive.

postpone_output
SYNTAX:  postpone_output size;
DEFAULT  1460
CONTEXT: http, server, location

If possible, the transmission of client data will be postponed until nginx has at least size bytes of data to send. The zero value disables postponing data transmission.

readAhead
SYNTAX:  read_ahead size;
DEFAULT  0
CONTEXT: http, server, location

Sets the amount of pre-reading for the kernel when working with file.
On Linux, the posix_fadvise(0, 0, 0, POSIX_FADV_SEQUENTIAL) system call is used, and so the size parameter is ignored.
On FreeBSD, the fcntl(O_READAHEAD, size) system call, supported since FreeBSD 9.0-CURRENT, is used. FreeBSD 7 has to be patched.

recursive_error_pages
SYNTAX:  recursive_error_pages on | off;
DEFAULT  off
CONTEXT: http, server, location

Enables or disables doing several redirects using the error_page directive. The number of such redirects is limited.

request_pool_size
SYNTAX:  request_pool_size size;
DEFAULT  4k
CONTEXT: http, server

 Allows accurate tuning of per-request memory allocations. This directive has minimal impact on performance and should not generally be used.
reset_timedout_connection

**SYNTAX:** `reset_timedout_connection on | off;`

**DEFAULT:** `off`

**CONTEXT:** `http, server, location`

Enables or disables resetting timed out connections. The reset is performed as follows. Before closing a socket, the `SO_LINGER` option is set on it with a timeout value of 0. When the socket is closed, TCP RST is sent to the client, and all memory occupied by this socket is released. This helps avoid keeping an already closed socket with filled buffers in a `FIN_WAIT1` state for a long time.

It should be noted that timed out keep-alive connections are closed normally.

resolver

**SYNTAX:** `resolver address ...[valid=time] [ipv6=on|off];`

**DEFAULT:** `—`

**CONTEXT:** `http, server, location`

Configures name servers used to resolve names of upstream servers into addresses, for example:

```
resolver 127.0.0.1 [:1]:5353;
```

An address can be specified as a domain name or IP address, and an optional port (1.3.1, 1.2.2). If port is not specified, the port 53 is used. Name servers are queried in a round-robin fashion.

Before version 1.1.7, only a single name server could be configured. Specifying name servers using IPv6 addresses is supported starting from versions 1.3.1 and 1.2.2.

By default, nginx will look up both IPv4 and IPv6 addresses while resolving. If looking up of IPv6 addresses is not desired, the `ipv6=off` parameter can be specified.

Resolving of names into IPv6 addresses is supported starting from version 1.5.8.

By default, nginx caches answers using the TTL value of a response. An optional `valid` parameter allows overriding it:

```
resolver 127.0.0.1 [:1]:5353 valid=30s;
```

Before version 1.1.9, tuning of caching time was not possible, and nginx always cached answers for the duration of 5 minutes.
resolver_timeout
SYNTAX: resolver_timeout time;
DEFAULT 30s
CONTEXT: http, server, location
Sets a timeout for name resolution, for example:

\[
\text{resolver_timeout 5s;}
\]

root
SYNTAX: root path;
DEFAULT html
CONTEXT: http, server, location, if in location
Sets the root directory for requests. For example, with the following configuration

\[
\text{location /i/ {}
  \text{  root /data/w3;}
\text{}}
\]

The /data/w3/i/top.gif file will be sent in response to the “/i/top.gif” request.
The path value can contain variables, except $document_root and $realpath_root.
A path to the file is constructed by merely adding a URI to the value of the root directive. If a URI has to be modified, the alias directive should be used.

satisfy
SYNTAX: satisfy all | any;
DEFAULT all
CONTEXT: http, server, location

Allows access if all (all) or at least one (any) of the ngx_http_access_module, ngx_http_auth_basic_module or ngx_http_auth_request_module modules allow access.
Example:

\[
\text{location / {}
  \text{  satisfy any;}
  \text{  allow 192.168.1.0/32;}
  \text{  deny all;}
  \text{  auth_basic "closed site";
  auth_basic_user_file conf/htpasswd;}
\text{}}
\]
satisfy_any

SYNTAX:  satisfy_any on | off;
DEFAULT  off
CONTEXT: http, server, location

This directive has been replaced by the any parameter of the satisfy directive.

send_lowat

SYNTAX:  send_lowat size;
DEFAULT  0
CONTEXT: http, server, location

If the directive is set to a non-zero value, nginx will try to minimize the number of send operations on client sockets by using either NOTE_LOWAT flag of the kqueue method or the SO_SNDLOWAT socket option. In both cases the specified size is used.

This directive is ignored on Linux, Solaris, and Windows.

send_timeout

SYNTAX:  send_timeout time;
DEFAULT  60s
CONTEXT: http, server, location

Sets a timeout for transmitting a response to the client. A timeout is set only between two successive write operations, not for the transmission of the whole response. If a client does not receive anything within this time, a connection is closed.

sendfile

SYNTAX:  sendfile on | off;
DEFAULT  off
CONTEXT: http, server, location, if in location

Enables or disables the use of sendfile.

sendfile_max_chunk

SYNTAX:  sendfile_max_chunk size;
DEFAULT  0
CONTEXT: http, server, location

When set to a non-zero value, limits the amount of data that can be transferred in a single sendfile call. Without the limit, one fast connection may seize the worker process entirely.
server

SYNTAX: server {...}

DEFAULT —

CONTEXT: http

Sets configuration for a virtual server. There is no clear separation between
IP-based (based on the IP address) and name-based (based on the Host request
header field) virtual servers. Instead, the listen directives describe all addresses
and ports that should accept connections for the server, and the server_name
directive lists all server names. Example configurations are provided in the
“How nginx processes a request” document.

server_name

SYNTAX: server_name name ...;

DEFAULT ""

CONTEXT: server

Sets names of a virtual server, for example:

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

The first name becomes the primary server name.
Server names can include an asterisk ("*”) replacing the first or last part
of a name:

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

Such names are called wildcard names.
The first two of the names mentioned above can be combined in one:

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

It is also possible to use regular expressions in server names, preceding the
name with a tilde ("~"):

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

Regular expressions can contain captures (0.7.40) that can later be used in
other directives:

```
server {
    server_name ~(www\.)?(.+)$;
    location / {
    }
```
Named captures in regular expressions create variables (0.8.25) that can later be used in other directives:

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

If the directive’s parameter is set to “$hostname” (0.9.4), the machine’s hostname is inserted.

It is also possible to specify an empty server name (0.7.11):

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

It allows this server to process requests without the Host header field — instead of the default server — for the given address:port pair. This is the default setting.

Before 0.8.48, the machine’s hostname was used by default.

During searching for a virtual server by name, if the name matches more than one of the specified variants, (e.g. both a wildcard name and regular expression match), the first matching variant will be chosen, in the following order of priority:

1. the exact name
2. the longest wildcard name starting with an asterisk, e.g. “*.example.com”
3. the longest wildcard name ending with an asterisk, e.g. “mail.*”
4. the first matching regular expression (in order of appearance in the configuration file)

Detailed description of server names is provided in a separate Server names document.

server_name_in_redirect

SYNTAX: server_name_in_redirect on | off;
DEFAULT off
CONTEXT: http, server, location

Enables or disables the use of the primary server name, specified by the server_name directive, in redirects issued by nginx. When the use of the primary server name is disabled, the name from the Host request header field is used. If this field is not present, the IP address of the server is used.

The use of a port in redirects is controlled by the port_in_redirect directive.

server_names_hash_bucket_size

SYNTAX: server_names_hash_bucket_size size;
DEFAULT 32|64|128
CONTEXT: http

Sets the bucket size for the server names hash tables. The default value depends on the size of the processor’s cache line. The details of setting up hash tables are provided in a separate document.

server_names_hash_max_size

SYNTAX: server_names_hash_max_size size;
DEFAULT 512
CONTEXT: http

Sets the maximum size of the server names hash tables. The details of setting up hash tables are provided in a separate document.

server_tokens

SYNTAX: server_tokens on | off;
DEFAULT on
CONTEXT: http, server, location

Enables or disables emitting nginx version in error messages and in the Server response header field.

tcp_nodelay

SYNTAX: tcp_nodelay on | off;
DEFAULT on
CONTEXT: http, server, location
Enables or disables the use of the **TCP_NODELAY** option. The option is enabled only when a connection is transitioned into the keep-alive state.

**tcp_nopush**

**SYNTAX:** tcp_nopush on | off;

**DEFAULT:** off

**CONTEXT:** http, server, location

Enables or disables the use of the **TCP_NOPUSH** socket option on FreeBSD or the **TCP_CORK** socket option on Linux. The options are enabled only when **sendfile** is used. Enabling the option allows

- sending the response header and the beginning of a file in one packet, on Linux and FreeBSD 4.*;
- sending a file in full packets.

**try_files**

**SYNTAX:** try_files file ... uri;

**SYNTAX:** try_files file ... = code;

**DEFAULT:**

**CONTEXT:** server, location

Checks the existence of files in the specified order and uses the first found file for request processing; the processing is performed in the current context. The path to a file is constructed from the **file** parameter according to the **root** and **alias** directives. It is possible to check directory’s existence by specifying a slash at the end of a name, e.g. “$uri/”. If none of the files were found, an internal redirect to the **uri** specified in the last parameter is made. For example:

```plaintext
location /images/ {
    try_files $uri /images/default.gif;
}

location = /images/default.gif {
    expires 30s;
}
```

The last parameter can also point to a named location, as shown in examples below. Starting from version 0.7.51, the last parameter can also be a **code**:

```plaintext
location / {
    try_files $uri $uri/index.html $uri.html =404;
}
```

Example in proxying Mongrel:
location / {
  try_files /system/maintenance.html $uri $uri/index.html $uri.html @mongrel;
}

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

Example for Drupal/FastCGI:

location / {
  try_files $uri $uri/ @drupal;
}

location ~ \..php$ {
  try_files $uri @drupal;
    fastcgi_pass ...;
    fastcgi_param SCRIPT_FILENAME /path/to$/fastcgi_script_name;
    fastcgi_param SCRIPT_NAME $fastcgi_script_name;
    fastcgi_param QUERY_STRING $args;
    ...
    other fastcgi_param's
}

location @drupal {
  fastcgi_pass ...;
    fastcgi_param SCRIPT_FILENAME /path/to/index.php;
    fastcgi_param SCRIPT_NAME /index.php;
    fastcgi_param QUERY_STRING q=$uri&$args;
    ...
    other fastcgi_param's
}

In the following example,

location / {
  try_files $uri $uri/ @drupal;
}

the try_files directive is equivalent to

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

And here,

location ~ \..php$ {
  try_files $uri @drupal;
    fastcgi_pass ...;
    fastcgi_param SCRIPT_FILENAME /path/to$/fastcgi_script_name;
    ...
}
try_files checks the existence of the PHP file before passing the request to the FastCGI server.

Example for Wordpress and Joomla:

```
location / {
    try_files $uri $uri/ @wordpress;
}
location ~ \..\php$ {
    try_files $uri @wordpress;
    fastcgi_pass ...;
    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;
    ... other fastcgi_param's
}
location @wordpress {
    fastcgi_pass ...;
    fastcgi_param SCRIPT_FILENAME /path/to/index.php;
    ... other fastcgi_param's
}
```

types

SYNTAX:  types { ... }
DEFAULT  text/html html; image/gif gif; image/jpeg jpg;
CONTEXT: http, server, location

Maps file name extensions to MIME types of responses. Extensions are case-insensitive. Several extensions can be mapped to one type, for example:

```
types {
    application/octet-stream bin exe dll;
    application/octet-stream deb;
    application/octet-stream dmg;
}
```

A sufficiently full mapping table is distributed with nginx in the conf/\¬mime.types file.

To make a particular location emit the “application/octet-stream” MIME type for all requests, the following configuration can be used:

```
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
Sets the bucket size for the types hash tables. The default value depends on the size of the processor’s cache line. The details of setting up hash tables are provided in a separate document.

**types_hash_max_size**

SYNTAX: `types_hash_max_size size;`

DEFAULT 1024

CONTEXT: http, server, location

Sets the maximum size of the types hash tables. The details of setting up hash tables are provided in a separate document.

**underscores_in_headers**

SYNTAX: `underscores_in_headers on | off;`

DEFAULT off

CONTEXT: http, server

Enables or disables the use of underscores in client request header fields. When the use of underscores is disabled, request header fields whose names contain underscores are marked as invalid and become subject to the `ignore_invalid_headers` directive.

If the directive is specified on the server level, its value is only used if a server is a default one. The value specified also applies to all virtual servers listening on the same address and port.

**variables_hash_bucket_size**

SYNTAX: `variables_hash_bucket_size size;`

DEFAULT 64

CONTEXT: http

Sets the bucket size for the variables hash table. The details of setting up hash tables are provided in a separate document.

**variables_hash_max_size**

SYNTAX: `variables_hash_max_size size;`

DEFAULT 512

CONTEXT: http

Sets the maximum size of the variables hash table. The details of setting up hash tables are provided in a separate document.

### 2.1.2 Embedded Variables

The `ngx_http_core_module` module supports embedded variables with names matching the Apache Server variables. First of all, these are variables...
representing client request header fields, such as $http_user_agent, $http_cookie, and so on. Also there are other variables:

\$arg_name
    argument name in the request line
\$args
    arguments in the request line
\$binary_remote_addr
    client address in a binary form, value’s length is always 4 bytes
\$body_bytes_sent
    number of bytes sent to a client, not counting the response header; this variable is compatible with the “%B” parameter of the mod_log_config Apache module
\$bytes_sent
    number of bytes sent to a client (1.3.8, 1.2.5)
\$connection
    connection serial number (1.3.8, 1.2.5)
\$connection_requests
    current number of requests made through a connection (1.3.8, 1.2.5)
\$content_length
    Content-Length request header field
\$content_type
    Content-Type request header field
\$cookie_name
    the name cookie
\$document_root
    root or alias directive’s value for the current request
\$document_uri
    same as \$uri
\$host
    in this order of precedence: host name from the request line, or host name from the Host request header field, or the server name matching a request
\$hostname
    host name
\$http_name
    arbitrary request header field; the last part of a variable name is the field name converted to lower case with dashes replaced by underscores
\$https
    “on” if connection operates in SSL mode, or an empty string otherwise
\$is_args
    “?” if a request line has arguments, or an empty string otherwise
\$limit_rate
    setting this variable enables response rate limiting; see limit_rate
\$msec
    current time in seconds with the milliseconds resolution (1.3.9, 1.2.6)
$nginx_version
nginx version

$pid
PID of the worker process

$pipe
“p” if request was pipelined, “.” otherwise (1.3.12, 1.2.7)

$proxy_protocol_addr
client address from the PROXY protocol header, or an empty string
otherwise (1.5.12)
The PROXY protocol must be previously enabled by setting the proxy_-
protocol parameter in the listen directive.

$proxy_string
same as $args

$realpath_root
an absolute pathname corresponding to the root or alias directive’s value
for the current request, with all symbolic links resolved to real paths

$remote_addr
client address

$remote_port
client port

$remote_user
user name supplied with the Basic authentication

$request
full original request line

$request_body
request body
The variable’s value is made available in locations processed by the
proxy_pass and fastcgi_pass directives.

$request_body_file
name of a temporary file with the request body
At the end of processing, the file needs to be removed. To always write
the request body to a file, client_body_in_file_only needs to be enabled.
When the name of a temporary file is passed in a proxied request or in a
request to a FastCGI server, passing the request body should be disabled
by the proxy_pass_request_body off and fastcgi_pass_request_body off
directives, respectively.

$request_completion
“OK” if a request has completed, or an empty string otherwise

$request_filename
file path for the current request, based on the root or alias directives,
and the request URI

$request_length
request length (including request line, header, and request body) (1.3.12,
1.2.7)

$request_method
request method, usually “GET” or “POST”

$\text{request\_time}$
request processing time in seconds with a milliseconds resolution (1.3.9, 1.2.6); time elapsed since the first bytes were read from the client

$\text{request\_uri}$
full original request URI (with arguments)

$\text{scheme}$
request scheme, “http” or “https”

$\text{sent\_http\_name}$
arbitrary response header field; the last part of a variable name is the field name converted to lower case with dashes replaced by underscores

$\text{server\_addr}$
an address of the server which accepted a request
Computing a value of this variable usually requires one system call. To avoid a system call, the listen directives must specify addresses and use the bind parameter.

$\text{server\_name}$
name of the server which accepted a request

$\text{server\_port}$
port of the server which accepted a request

$\text{server\_protocol}$
request protocol, usually “HTTP/1.0” or “HTTP/1.1”

$\text{status}$
response status (1.3.2, 1.2.2)

$\text{tcpinfo\_rtt}, \text{tcpinfo\_rttvar}, \text{tcpinfo\_snd\_cwnd}, \text{tcpinfo\_rcv\_space}$
information about the client TCP connection; available on systems that support the TCP_INFO socket option

$\text{time\_iso8601}$
local time in the ISO 8601 standard format (1.3.12, 1.2.7)

$\text{time\_local}$
local time in the Common Log Format (1.3.12, 1.2.7)

$\text{uri}$
current URI in request, normalized
The value of $\text{uri}$ may change during request processing, e.g. when doing internal redirects, or when using index files.
2.2 Module ngx_http_access_module

2.2.1 Summary

The ngx_http_access_module module allows limiting access to certain client addresses.

Access can also be limited by password or by the result of subrequest. Simultaneous limitation of access by address and by password is controlled by the satisfy directive.

2.2.2 Example Configuration

```conf
location / {
    deny 192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0:db8::/32;
    deny all;
}
```

The rules are checked in sequence until the first match is found. In this example, access is allowed only for IPv4 networks 10.1.1.0/16 and 192.168.1.0/24 excluding the address 192.168.1.1, and for IPv6 network 2001:0:db8::/32. In case of a lot of rules, the use of the ngx_http_geo_module module variables is preferable.

2.2.3 Directives

allow

**SYNTAX:** `allow address | CIDR | unix: | all;`

**DEFAULT —**

**CONTEXT:** http, server, location, limit_except

Allows access for the specified network or address. If the special value `unix:` is specified (1.5.1), allows access for all UNIX-domain sockets.

deny

**SYNTAX:** `deny address | CIDR | unix: | all;`

**DEFAULT —**

**CONTEXT:** http, server, location, limit_except

Denies access for the specified network or address. If the special value `unix:` is specified (1.5.1), denies access for all UNIX-domain sockets.
2.3 Module ngx_http_addition_module

2.3.1 Summary

The ngx_http_addition_module module is a filter that adds text before and after a response. This module is not built by default, it should be enabled with the --with-http_addition_module configuration parameter.

2.3.2 Example Configuration

```
location / {
    add_before_body /before_action;
    add_after_body /after_action;
}
```

2.3.3 Directives

**add_before_body**

SYNTAX: add_before_body uri;

DEFAULT —

CONTEXT: http, server, location

Adds the text returned as a result of processing a given subrequest before the response body. An empty string ("") as a parameter cancels addition inherited from the previous configuration level.

**add_after_body**

SYNTAX: add_after_body uri;

DEFAULT —

CONTEXT: http, server, location

Adds the text returned as a result of processing a given subrequest after the response body. An empty string ("") as a parameter cancels addition inherited from the previous configuration level.

**addition_types**

SYNTAX: addition_types mime-type ...;

DEFAULT text/html

CONTEXT: http, server, location

This directive appeared in version 0.7.9.

Allows adding text in responses with the specified MIME types, in addition to "text/html". The special value "*" matches any MIME type (0.8.29).
2.4 Module ngx_http_auth_basic_module

2.4.1 Summary

The ngx_http_auth_basic_module module allows limiting access to resources by validating the user name and password using the “HTTP Basic Authentication” protocol.

Access can also be limited by address or by the result of subrequest. Simultaneous limitation of access by address and by password is controlled by the satisfy directive.

2.4.2 Example Configuration

```plaintext
location / {
    auth_basic "closed site";
    auth_basic_user_file conf/htpasswd;
}
```

2.4.3 Directives

**auth_basic**

- **SYNTAX:** `auth_basic string | off;`
- **DEFAULT:** `off`
- **CONTEXT:** http, server, location, limit_except

Enables validation of user name and password using the “HTTP Basic Authentication” protocol. The specified parameter is used as a realm. Parameter value can contain variables (1.3.10, 1.2.7). The special value `off` allows cancelling the effect of the auth_basic directive inherited from the previous configuration level.

**auth_basic_user_file**

- **SYNTAX:** `auth_basic_user_file file;`
- **DEFAULT:** `—`
- **CONTEXT:** http, server, location, limit_except

Specifies a file that keeps user names and passwords, in the following format:

```
# comment
name1:password1
name2:password2:comment
name3:password3
```

The following password types are supported:

- encrypted with the `crypt` function; can be generated using the “htpasswd” utility from the Apache HTTP Server distribution or the “openssl passwd” command;
• hashed with the Apache variant of the MD5-based password algorithm (apr1); can be generated with the same tools;

• specified by the “{scheme} data” syntax (1.0.3+) as described in RFC 2307; currently implemented schemes include PLAIN (an example one, should not be used), SHA (1.3.13) (plain SHA-1 hashing, should not be used) and SSHA (salted SHA-1 hashing, used by some software packages, notably OpenLDAP and Dovecot).

Support for SHA scheme was added only to aid in migration from other web servers. It should not be used for new passwords, since unsalted SHA-1 hashing that it employs is vulnerable to rainbow table attacks.
2.5 Module ngx_http_auth_request_module

2.5.1 Summary

The ngx_http_auth_request_module module (1.5.4+) implements client authorization based on the result of a subrequest. If the subrequest returns a 2xx response code, the access is allowed. If it returns 401 or 403, the access is denied with the corresponding error code. Any other response code returned by the subrequest is considered an error.

For the 401 error, the client also receives the WWW-Authenticate header from the subrequest response.

This module is not built by default, it should be enabled with the --with-http_auth_request_module configuration parameter.

The module may be combined with other access modules, such as ngx_http_access_module and ngx_http_auth_basic_module, via the satisfy directive.

Currently, responses to authorization subrequests cannot be cached (using proxy_cache, proxy_store, etc.).

2.5.2 Example Configuration

```
location /private/ {
   auth_request /auth;
   ...
}

location = /auth {
   proxy_pass ...;
   proxy_pass_request_body off;
   proxy_set_header Content-Length "";
   proxy_set_header X-Original-URI $request_uri;
}
```

2.5.3 Directives

auth_request

SYNTAX: auth_request uri | off;
DEFINATE off
CONTEXT: http, server, location

Enables authorization based on the result of a subrequest and sets the URI to which the subrequest will be sent.

auth_request_set

SYNTAX: auth_request_set variable value;
DEFINATE —
CONTEXT: http, server, location
Sets the request variable to the given value after the authorization request completes. The value may contain variables from the authorization request, such as $upstream_http_*.
2.6 Module ngx_http_autoindex_module

2.6.1 Summary

The ngx_http_autoindex_module module processes requests ending with the slash character (’/’) and produces a directory listing. Usually a request is passed to the ngx_http_autoindex_module module when the ngx_http-index_module module cannot find an index file.

2.6.2 Example Configuration

```plaintext
location / {
  autoindex on;
}
```

2.6.3 Directives

**autoindex**

SYNTAX: autoindex on | off;
DEFAULT off
CONTEXT: http, server, location

Enables or disables the directory listing output.

**autoindex_exact_size**

SYNTAX: autoindex_exact_size on | off;
DEFAULT on
CONTEXT: http, server, location

Specifies whether exact file sizes should be output in the directory listing, or rather rounded to kilobytes, megabytes, and gigabytes.

**autoindex_localtime**

SYNTAX: autoindex_localtime on | off;
DEFAULT off
CONTEXT: http, server, location

Specifies whether times in the directory listing should be output in the local time zone or UTC.
2.7 Module ngx_http_browser_module

2.7.1 Summary

The ngx_http_browser_module module creates variables whose values depend on the value of the User-Agent request header field:

\$modern_browser 
equalsthe value set by the modern_browser_value directive, if a browser was identified as modern;

\$ancient_browser 
equalsthe value set by the ancient_browser_value directive, if a browser was identified as ancient;

\$msie 
equalsts "1" if a browser was identified as MSIE of any version.

2.7.2 Example Configuration

Choosing an index file:

```bash
modern_browser_value "modern.";
modern_browser msie 5.5;
modern_browser gecko 1.0.0;
modern_browser opera 9.0;
modern_browser safari 413;
modern_browser konqueror 3.0;
index index.$(modern_browser)html index.html;
```

Redirection for old browsers:

```bash
modern_browser msie 5.0;
modern_browser gecko 0.9.1;
modern_browser opera 8.0;
modern_browser safari 413;
modern_browser konqueror 3.0;
modern_browser unlisted;
ancient_browser Links Lynx netscape4;
if ($ancient_browser) {
    rewrite " /ancient.html;
}
```

2.7.3 Directives

**ancient_browser**

SYNTAX: ancient_browser string ...;

DEFAULT —

CONTEXT: http, server, location
If any of the specified substrings is found in the *User-Agent* request header field, the browser will be considered ancient. The special string “netscape4” corresponds to the regular expression “Mozilla/[1-4]”.

**ancient_browser_value**

```
SYNTAX:  ancient_browser_value string;
DEFAULT  1
CONTEXT: http, server, location
```

Sets a value for the $ancient_browser$ variables.

**modern_browser**

```
SYNTAX:  modern_browser browser version;
SYNTAX:  modern_browser unlisted;
DEFAULT  —
CONTEXT: http, server, location
```

Specifies a version starting from which a browser is considered modern. A browser can be any one of the following: *msie*, *gecko* (browsers based on Mozilla), *opera*, *safari*, or *konqueror*.

Versions can be specified in the following formats: X, X.X, X.X.X, or X.X.X.X. The maximum values for each of the format are 4000, 4000.99, 4000.99.99, and 4000.99.99.99, respectively.

The special value *unlisted* specifies to consider a browser as modern if it was not listed by the *modern_browser* and *ancient_browser* directives. Otherwise such a browser is considered ancient. If a request does not provide the *User-Agent* field in the header, the browser is treated as not being listed.

**modern_browser_value**

```
SYNTAX:  modern_browser_value string;
DEFAULT  1
CONTEXT: http, server, location
```

Sets a value for the $modern_browser$ variables.
2.8 Module ngx_http_charset_module

2.8.1 Summary

The ngx_http_charset_module module adds the specified charset to the `Content-Type` response header field. In addition, the module can convert data from one charset to another, with some limitations:

- conversion is performed one way — from server to client,
- only single-byte charsets can be converted
- or single-byte charsets to/from UTF-8.

2.8.2 Example Configuration

```plaintext
include conf/koi-win;

charset windows-1251;
source_charset koi8-r;
```

2.8.3 Directives

**charset**

**SYNTAX:** `charset charset | off;`

**DEFAULT** `off`

**CONTEXT:** http, server, location, if in location

Adds the specified charset to the `Content-Type` response header field. If this charset is different from the charset specified in the `source_charset` directive, a conversion is performed.

The parameter `off` cancels the addition of charset to the `Content-Type` response header field.

A charset can be defined with a variable:

```plaintext
charset $charset;
```

In such a case, all possible values of a variable need to be present in the configuration at least once in the form of the `charset_map`, `charset`, or `source_charset` directives. For utf-8, windows-1251, and koi8-r charsets, it is sufficient to include the files `conf/koi-win`, `conf/koi-utf`, and `conf/win-utf` into configuration. For other charsets, simply making a fictitious conversion table works, for example:

```plaintext
charset_map iso-8859-5 _ ( )
```

In addition, a charset can be set in the `X-Accel-Charset` response header field. This capability can be disabled using the `proxy_ignore_headers` and `fastcgi_ignore_headers` directives.
CHAPTER 2. HTTP SERVER MODULES  2.8. MODULE NGX_HTTP_CHARSET_MODULE

charset_map

SYNTAX:  

```
charset_map charset1 charset2 { ... }
```

DEFAULT —

CONTEXT: http

Describes the conversion table from one charset to another. A reverse conversion table is built using the same data. Character codes are given in hexadecimal. Missing characters in the range 80-FF are replaced with “?”.

When converting from UTF-8, characters missing in a one-byte charset are replaced with “&#XXXX;”.

Example:

```perl
charset_map koi8-r windows-1251 {
  C0 FE ; # small yu
  C1 E0 ; # small a
  C2 E1 ; # small b
  C3 F6 ; # small ts
  ...
}
```

When describing a conversion table to UTF-8, codes for the UTF-8 charset should be given in the second column, for example:

```perl
charset_map koi8-r utf-8 {
  C0 D18E ; # small yu
  C1 D0B0 ; # small a
  C2 D0B1 ; # small b
  C3 D186 ; # small ts
  ...
}
```

Full conversion tables from koi8-r to windows-1251, and from koi8-r and windows-1251 to utf-8 are provided in the distribution files conf/koi-win, conf/koi-utf, and conf/win-utf.

charset_types

SYNTAX:  

```
charset_types mime-type ...;
```

DEFAULT text/html text/xml text/plain text/vnd.wap.wml
application/javascript application/rss+xml

CONTEXT: http, server, location

This directive appeared in version 0.7.9.

Enables module processing in responses with the specified MIME types in addition to “text/html”. The special value “*” matches any MIME type (0.8.29).

Until version 1.5.4, “application/x-javascript” was used as the default MIME type instead of “application/javascript”. 

Nginx, Inc.  p.73 of 242
override_charset

SYNTAX: override_charset on | off;
DEFAULT off
CONTEXT: http, server, location, if in location

Determines whether a conversion should be performed for answers received from a proxied or FastCGI server when the answers already carry a charset in the Content-Type response header field. If conversion is enabled, a charset specified in the received response is used as a source charset.

It should be noted that if a response is received in a subrequest then the conversion from the response charset to the main request charset is always performed, regardless of the override_charset directive setting.

source_charset

SYNTAX: source_charset charset;
DEFAULT —
CONTEXT: http, server, location, if in location

Defines the source charset of a response. If this charset is different from the charset specified in the charset directive, a conversion is performed.
2.9 Module ngx_http_dav_module

2.9.1 Summary

The ngx_http_dav_module module is intended for file management automation via the WebDAV protocol. The module processes HTTP and WebDAV methods PUT, DELETE, MKCOL, COPY, and MOVE.

This module is not built by default, it should be enabled with the --with-http_dav_module configuration parameter.

WebDAV clients that require additional WebDAV methods to operate will not work with this module.

2.9.2 Example Configuration

```
location / {
  root / data / www;
  client_body_temp_path /data/client_temp;
  dav_methods PUT DELETE MKCOL COPY MOVE;
  create_full_put_path on;
  dav_access group:rw all:r;
  limit_except GET {
    allow 192.168.1.0/32;
    deny all;
  }
}
```

2.9.3 Directives

dav_access

SYNTAX: dav_access users:permissions ...;

DEFAULT user:rw

CONTEXT: http, server, location

Sets access permissions for newly created files and directories, e.g.:

```
dav_access user:rw group:rw all:r;
```

If any group or all access permissions are specified then user permissions may be omitted:

```
dav_access group:rw all:r;
```
**dav_methods**

SYNTAX: `dav_methods off | method ...;`

DEFAULT: `off`

CONTEXT: `http, server, location`

Allows the specified HTTP and WebDAV methods. The parameter `off` denies all methods processed by this module. The following methods are supported: `PUT`, `DELETE`, `MKCOL`, `COPY`, and `MOVE`.

A file uploaded with the PUT method is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the persistent store can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both saved files and a directory holding temporary files, set by the `client_body_temp_path` directive, are put on the same file system.

When creating a file with the PUT method, it is possible to specify the modification date by passing it in the `Date` header field.

**create_full_put_path**

SYNTAX: `create_full_put_path on | off;`

DEFAULT: `off`

CONTEXT: `http, server, location`

The WebDAV specification only allows creating files in already existing directories. This directive allows creating all needed intermediate directories.

**min_delete_depth**

SYNTAX: `min_delete_depth number;`

DEFAULT: `0`

CONTEXT: `http, server, location`

Allows the DELETE method to remove files provided that the number of elements in a request path is not less than the specified number. For example, the directive

```
min_delete_depth 4;
```

allows removing files on requests

```
/users/00/00/name
/users/00/00/name/pic.jpg
/users/00/00/page.html
```

and denies the removal of

```
/users/00/00
```
2.10 Module ngx_http_empty_gif_module

2.10.1 Summary

The ngx_http_empty_gif_module module emits single-pixel transparent GIF.

2.10.2 Example Configuration

```
location = /_.gif {
  empty_gif;
}
```

2.10.3 Directives

empty_gif

SYNTAX:   empty_gif;
DEFAULT —
CONTEXT: location

Turns on module processing in a surrounding location.
2.11 Module ngx_http_fastcgi_module

2.11.1 Summary

The ngx_http_fastcgi_module module allows passing requests to a FastCGI server.

2.11.2 Example Configuration

```
location / {
    fastcgi_pass localhost:9000;
    fastcgi_index index.php;

    fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
    fastcgi_param QUERY_STRING $query_string;
    fastcgi_param REQUEST_METHOD $request_method;
    fastcgi_param CONTENT_TYPE $content_type;
    fastcgi_param CONTENT_LENGTH $content_length;
}
```

2.11.3 Directives

**fastcgi_bind**

**SYNTAX:** `fastcgi_bind address | off;`

**DEFAULT** —

**CONTEXT:** `http, server, location`

This directive appeared in version 0.8.22.

Makes outgoing connections to a FastCGI server originate from the specified local IP `address`. Parameter value can contain variables (1.3.12). The special value `off` (1.3.12) cancels the effect of the `fastcgi_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address.

**fastcgi_buffer_size**

**SYNTAX:** `fastcgi_buffer_size size;`

**DEFAULT** `4k|8k`

**CONTEXT:** `http, server, location`

Sets the `size` of the buffer used for reading the first part of a response received from the FastCGI server. This part usually contains a small response header. By default, the buffer size is equal to the size of one buffer set by the `fastcgi_buffers` directive. It can be made smaller however.
fastcgi_buffering

**SYNTAX:** `fastcgi_buffering on | off;`
**DEFAULT** `on`
**CONTEXT:** `http, server, location`

This directive appeared in version 1.5.6.

Enables or disables buffering of responses from the FastCGI server.

When buffering is enabled, nginx receives a response from the FastCGI server as soon as possible, saving it into the buffers set by the `fastcgi_buffer_size` and `fastcgi_buffers` directives. If the whole response does not fit into memory, a part of it can be saved to a temporary file on the disk. Writing to temporary files is controlled by the `fastcgi_max_temp_file_size` and `fastcgi_temp_file_write_size` directives.

When buffering is disabled, a response is passed to a client synchronously, immediately as it is received. nginx will not try to read the whole response from the FastCGI server. The maximum size of the data that nginx can receive from the server at a time is set by the `fastcgi_buffer_size` directive.

Buffering can also be enabled or disabled by passing “yes” or “no” in the `X-Accel-Buffering` response header field. This capability can be disabled using the `fastcgi_ignore_headers` directive.

fastcgi_buffers

**SYNTAX:** `fastcgi_buffers number size;`
**DEFAULT** `8 4k|8k`
**CONTEXT:** `http, server, location`

Sets the `number` and `size` of buffers used for reading a response from the FastCGI server, for a single connection. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

fastcgi_busy_buffers_size

**SYNTAX:** `fastcgi_busy_buffers_size size;`
**DEFAULT** `8k|16k`
**CONTEXT:** `http, server, location`

When buffering of responses from the FastCGI server is enabled, limits the total `size` of buffers that can be busy sending a response to the client while the response is not yet fully read. In the mean time, the rest of the buffers can be used for reading a response and, if needed, buffering part of a response to a temporary file. By default, `size` is limited by the size of two buffers set by the `fastcgi_buffer_size` and `fastcgi_buffers` directives.

fastcgi_cache

**SYNTAX:** `fastcgi_cache zone | off;`
**DEFAULT** `off`
**CONTEXT:** `http, server, location`
Defines a shared memory zone used for caching. The same zone can be used in several places. The off parameter disables caching inherited from the previous configuration level.

**fastcgi_cache_bypass**

**SYNTAX:** `fastcgi_cache_bypass string...`

**DEFAULT** —

**CONTEXT:** http, server, location

Defines conditions under which the response will not be taken from a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be taken from the cache:

```plaintext
fastcgi_cache_bypass $cookie_nocache $arg_nocache $arg_comment ;
fastcgi_cache_bypass $http_pragma $http_authorization ;
```

Can be used along with the **fastcgi_no_cache** directive.

**fastcgi_cache_key**

**SYNTAX:** `fastcgi_cache_key string;`

**DEFAULT** —

**CONTEXT:** http, server, location

Defines a key for caching, for example

```plaintext
fastcgi_cache_key localhost:9000$request_uri ;
```

**fastcgi_cache_lock**

**SYNTAX:** `fastcgi_cache_lock on | off;`

**DEFAULT** off

**CONTEXT:** http, server, location

This directive appeared in version 1.1.12.

When enabled, only one request at a time will be allowed to populate a new cache element identified according to the **fastcgi_cache_key** directive by passing a request to a FastCGI server. Other requests of the same cache element will either wait for a response to appear in the cache or the cache lock for this element to be released, up to the time set by the **fastcgi_cache_lock_timeout** directive.

**fastcgi_cache_lock_timeout**

**SYNTAX:** `fastcgi_cache_lock_timeout time;`

**DEFAULT** 5s

**CONTEXT:** http, server, location

This directive appeared in version 1.1.12.
Sets a timeout for fastcgi_cache_lock.

**fastcgi_cache_methods**

**SYNTAX:**
```
fastcgi_cache_methods GET | HEAD | POST ...;
```

**DEFAULT**
```
GET HEAD
```

**CONTEXT:** http, server, location

This directive appeared in version 0.7.59.

If the client request method is listed in this directive then the response will be cached. “GET” and “HEAD” methods are always added to the list, though it is recommended to specify them explicitly. See also the fastcgi_no_cache directive.

**fastcgi_cache_min_uses**

**SYNTAX:**
```
fastcgi_cache_min_uses number;
```

**DEFAULT**
```
1
```

**CONTEXT:** http, server, location

Sets the number of requests after which the response will be cached.

**fastcgi_cache_path**

**SYNTAX:**
```
fastcgi_cache_path path [levels=levels] [keys_zone=name:size]
[inactive=time] [max_size=size] [loader_files=number]
[loader_sleep=time] [loader_threshold=time];
```

**DEFAULT**
```
—
```

**CONTEXT:** http

Sets the path and other parameters of a cache. Cache data are stored in files. Both the key and file name in a cache are a result of applying the MD5 function to the proxied URL.

The levels parameter defines hierarchy levels of a cache. For example, in the following configuration

```
fastcgi_cache_path /data/nginx/cache levels=1:2 keys_zone=one:10m;
```

file names in a cache will look like this:

```
/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

A cached response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the cache can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both cache and a directory holding temporary files, set by the fastcgi_temp_path directive, are put on the same file system.
In addition, all active keys and information about data are stored in a shared memory zone, whose name and size are configured by the `keys_zone` parameter. Cached data that are not accessed during the time specified by the `inactive` parameter get removed from the cache regardless of their freshness. By default, `inactive` is set to 10 minutes.

The special “cache manager” process monitors the maximum cache size set by the `max_size` parameter. When this size is exceeded, it removes the least recently used data.

A minute after the start the special “cache loader” process is activated. It loads information about previously cached data stored on file system into a cache zone. The loading is done in iterations. During one iteration no more than `loader_files` items are loaded (by default, 100). Besides, the duration of one iteration is limited by the `loader_threshold` parameter (by default, 200 milliseconds). Between iterations, a pause configured by the `loader_sleep` parameter (by default, 50 milliseconds) is made.

```plaintext
fastcgi_cache_purge
SYNTAX:    fastcgi_cache_purge string ...
DEFAULT —
CONTEXT:   http, server, location
This directive appeared in version 1.5.7.

Defines conditions under which the request will be considered a cache purge request. If at least one value of the string parameters is not empty and is not equal to “0” then the cache entry with a corresponding cache key is removed. The result of successful operation is indicated by returning the 204 No Content response.

If the cache key of a purge request ends with an asterisk (“*”), all cache entries matching the wildcard key will be removed from the cache.

Example configuration:

```plaintext
fastcgi_cache_path /data/nginx/cache keys_zone=cache_zone:10m;
map $request_method $purge_method {
    PURGE 1;
    default 0;
}
server {
...
    location / {
        fastcgi_pass backend;
        fastcgi_cache_cache_zone;
        fastcgi_cache_key $uri;
        fastcgi_cache_purge $purge_method;
    }
}
```

This functionality is available as part of our commercial subscription.
fastcgi_cache_revalidate

SYNTAX:  fastcgi_cache_revalidate on | off;
DEFAULT  off
CONTEXT: http, server, location
This directive appeared in version 1.5.7.

Enables revalidation of expired cache items using conditional requests with the If-Modified-Since header field.

fastcgi_cache_use_stale

SYNTAX:  fastcgi_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_503 | http_403 | http_404 | off . . . ;
DEFAULT  off
CONTEXT: http, server, location

Determines in which cases a stale cached response can be used when an error occurs during communication with the FastCGI server. The directive’s parameters match the parameters of the fastcgi_next_upstream directive.

Additionally, the updating parameter permits using a stale cached response if it is currently being updated. This allows minimizing the number of accesses to FastCGI servers when updating cached data.

To minimize the number of accesses to FastCGI servers when populating a new cache element, the fastcgi_cache_lock directive can be used.

fastcgi_cache_valid

SYNTAX:  fastcgi_cache_valid [code . . .] time;
DEFAULT —
CONTEXT: http, server, location

Sets caching time for different response codes. For example, the following directives

```
fastcgi_cache_valid 200 302 10m;
fastcgi_cache_valid 404 1m;
```

set 10 minutes of caching for responses with codes 200 and 302 and 1 minute for responses with code 404.

If only caching time is specified

```
fastcgi_cache_valid 5m;
```

then only 200, 301, and 302 responses are cached.

In addition, the any parameter can be specified to cache any responses:

```
fastcgi_cache_valid 200 302 10m;
fastcgi_cache_valid 301 1h;
fastcgi_cache_valid any 1m;
```
Parameters of caching can also be set directly in the response header. This has higher priority than setting of caching time using the directive. The \textit{X-Accel-Expires} header field sets caching time of a response in seconds. The zero value disables caching for a response. If a value starts with the \@ prefix, it sets an absolute time in seconds since Epoch, up to which the response may be cached. If header does not include the \textit{X-Accel-Expires} field, parameters of caching may be set in the header fields \textit{Expires} or \textit{Cache-Control}. If a header includes the \textit{Set-Cookie} field, such a response will not be cached. Processing of one or more of these response header fields can be disabled using the \texttt{fastcgi-ignore_headers} directive.

\texttt{fastcgi\_catch\_stderr}

\begin{verbatim}
SYNTAX: \texttt{fastcgi\_catch\_stderr string.}
DEFAULT —
CONTEXT: http, server, location

Sets a string to search for in the error stream of a response received from a FastCGI server. If the \textit{string} is found then it is considered that the FastCGI server has returned an invalid response. This allows handling application errors in nginx, for example:

\begin{verbatim}
location /php {
    fastcgi_pass backend:9000;
    ...
    fastcgi\_catch\_stderr "PHP Fatal error";
    fastcgi\_next\_upstream error timeout invalid\_header;
}
\end{verbatim}
\end{verbatim}

\texttt{fastcgi\_connect\_timeout}

\begin{verbatim}
SYNTAX: \texttt{fastcgi\_connect\_timeout time;}
DEFAULT 60s
CONTEXT: http, server, location

Defines a timeout for establishing a connection with a FastCGI server. It should be noted that this timeout cannot usually exceed 75 seconds.
\end{verbatim}

\texttt{fastcgi\_hide\_header}

\begin{verbatim}
SYNTAX: \texttt{fastcgi\_hide\_header field;}
DEFAULT —
CONTEXT: http, server, location

By default, nginx does not pass the header fields \textit{Status} and \textit{X-Accel-...} from the response of a FastCGI server to a client. The \texttt{fastcgi\_hide\_header} directive sets additional fields that will not be passed. If, on the contrary, the passing of fields needs to be permitted, the \texttt{fastcgi\_pass\_header} directive can be used.
\end{verbatim}
fastcgi_ignore_client_abort

SYNTAX:  fastcgi_ignore_client_abort on | off;
DEFAULT  off
CONTEXT: http, server, location

Determines whether the connection with a FastCGI server should be closed when a client closes a connection without waiting for a response.

fastcgi_ignore_headers

SYNTAX:  fastcgi_ignore_headers field ...;
DEFAULT —
CONTEXT: http, server, location

Disables processing of certain response header fields from the FastCGI server. The following fields can be ignored: X-Accel-Redirect, X-Accel-Expires, X-Accel-Limit-Rate (1.1.6), X-Accel-Buffering (1.1.6), X-Accel-Charset (1.1.6), Expires, Cache-Control, and Set-Cookie (0.8.44).

If not disabled, processing of these header fields has the following effect:

- X-Accel-Expires, Expires, Cache-Control, and Set-Cookie set the parameters of response caching;
- X-Accel-Redirect performs an internal redirect to the specified URI;
- X-Accel-Limit-Rate sets the rate limit for transmission of a response to a client;
- X-Accel-Buffering enables or disables buffering of a response;
- X-Accel-Charset sets the desired charset of a response.

fastcgi_index

SYNTAX:  fastcgi_index name;
DEFAULT —
CONTEXT: http, server, location

Sets a file name that will be appended after a URI that ends with a slash, in the value of the $fastcgi_script_name variable. For example, with these settings

```
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
```

and the “/page.php” request, the SCRIPT_FILENAME parameter will be equal to “/home/www/scripts/php/page.php”, and with the “/” request it will be equal to “/home/www/scripts/php/index.php”.

Nginx, Inc.  p.85 of 242
fastcgi_intercept_errors
  SYNTAX:  fastcgi_intercept_errors on | off;
  DEFAULT  off
  CONTEXT:  http, server, location

  Determines whether FastCGI server responses with codes greater than or equal to 300 should be passed to a client or be redirected to nginx for processing with the error_page directive.

fastcgi_keep_conn
  SYNTAX:  fastcgi_keep_conn on | off;
  DEFAULT  off
  CONTEXT:  http, server, location

  This directive appeared in version 1.1.4.

  By default, a FastCGI server will close a connection right after sending the response. However, when this directive is set to the value on, nginx will instruct a FastCGI server to keep connections open. This is necessary, in particular, for keepalive connections to FastCGI servers to function.

fastcgi_max_temp_file_size
  SYNTAX:  fastcgi_max_temp_file_size size;
  DEFAULT  1024m
  CONTEXT:  http, server, location

  When buffering of responses from the FastCGI server is enabled, and the whole response does not fit into the memory buffers set by the fastcgi_buffer_size and fastcgi_buffers directives, a part of the response can be saved to a temporary file. This directive sets the maximum size of a temporary file. The size of data written to a temporary file at a time is set by the fastcgi_temp_file_write_size directive.

  The zero value disables buffering of responses to temporary files.

fastcgi_next_upstream
  SYNTAX:  fastcgi_next_upstream error | timeout | invalid_header | http_500
  | http_503 | http_403 | http_404 | off ...
  DEFAULT  error timeout
  CONTEXT:  http, server, location

  Specifies in which cases a request should be passed to the next server:

  error
  an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

  timeout
  a timeout has occurred while establishing a connection with the server, passing a request to it, or reading the response header;
invalid_header
  a server returned an empty or invalid response;
http_500
  a server returned a response with the code 500;
http_503
  a server returned a response with the code 503;
http_403
  a server returned a response with the code 403;
http_404
  a server returned a response with the code 404;
off
  disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an unsuccessful attempt of communication with a server. The cases of error, timeout and invalid_header are always considered unsuccessful attempts, even if they are not specified in the directive. The cases of http_500 and http_503 are considered unsuccessful attempts only if they are specified in the directive. The cases of http_403 and http_404 are never considered unsuccessful attempts.

fastcgi_no_cache

SYNTAX:  fastcgi_no_cache string ...;
DEFAULT —
CONTEXT: http, server, location

Defines conditions under which the response will not be saved to a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be saved:

```bash
fastcgi_no_cache $cookie_nocache $arg_nocache $arg_comment;
fastcgi_no_cache $http_pragma $http_authorization;
```

Can be used along with the fastcgi_cache_bypass directive.

fastcgi_param

SYNTAX:  fastcgi_param parameter value [if_not_empty];
DEFAULT —
CONTEXT: http, server, location

Sets a parameter that should be passed to the FastCGI server. A value can contain text, variables, and their combination. These directives are inherited from the previous level if and only if there are no fastcgi_param directives defined on the current level.

The following example shows the minimum required settings for PHP:
The `SCRIPT_FILENAME` parameter is used in PHP for determining the script name, and the `QUERY_STRING` parameter is used to pass request parameters.

For scripts that process `POST` requests, the following three parameters are also required:

```perl
fastcgi_param REQUEST_METHOD $request_method;
fastcgi_param CONTENT_TYPE $content_type;
fastcgi_param CONTENT_LENGTH $content_length;
```

If PHP was built with the `--enable-force-cgi-redirect` configuration parameter, the `REDIRECT_STATUS` parameter should also be passed with the value “200”:

``` perl
fastcgi_param REDIRECT_STATUS 200;
```

If a directive is specified with `if_not_empty` (1.1.11) then such a parameter will not be passed to the server until its value is not empty:

``` perl
fastcgi_param HTTPS $https if_not_empty;
```

`fastcgi_pass`

**SYNTAX:** `fastcgi_pass address;`

**DEFAULT** —

**CONTEXT:** location, if in location

Sets the address of a FastCGI server. The address can be specified as a domain name or IP address, and an optional port:

``` perl
fastcgi_pass localhost:9000;
```

or as a UNIX-domain socket path:

``` perl
fastcgi_pass unix:/tmp/fastcgi.socket;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a server group.

`fastcgi_pass_header`

**SYNTAX:** `fastcgi_pass_header field;`

**DEFAULT** —

**CONTEXT:** http, server, location

Permits passing otherwise disabled header fields from a FastCGI server to a client.
fastcgi_read_timeout

SYNTAX:  fastcgi_read_timeout time;
DEFAULT 60s
CONTEXT: http, server, location

Defines a timeout for reading a response from the FastCGI server. A timeout is set only between two successive read operations, not for the transmission of the whole response. If a FastCGI server does not transmit anything within this time, a connection is closed.

fastcgi_pass_request_body

SYNTAX:  fastcgi_pass_request_body on | off;
DEFAULT on
CONTEXT: http, server, location

Indicates whether the original request body is passed to the FastCGI server. See also the fastcgi_pass_request_headers directive.

fastcgi_pass_request_headers

SYNTAX:  fastcgi_pass_request_headers on | off;
DEFAULT on
CONTEXT: http, server, location

Indicates whether the header fields of the original request are passed to the FastCGI server. See also the fastcgi_pass_request_body directive.

fastcgi_send_lowat

SYNTAX:  fastcgi_send_lowat size;
DEFAULT 0
CONTEXT: http, server, location

If the directive is set to a non-zero value, nginx will try to minimize the number of send operations on outgoing connections to a FastCGI server by using either NOTE_LOWAT flag of the kqueue method, or the SO_SNDLOWAT socket option, with the specified size.

This directive is ignored on Linux, Solaris, and Windows.

fastcgi_send_timeout

SYNTAX:  fastcgi_send_timeout time;
DEFAULT 60s
CONTEXT: http, server, location

Sets a timeout for transmitting a request to the FastCGI server. A timeout is set only between two successive write operations, not for the transmission of the whole request. If a FastCGI server does not receive anything within this time, a connection is closed.
fastcgi_split_path_info

**SYNTAX:**  fastcgi_split_path_info regex;

**DEFAULT** —

**CONTEXT:** location

Defines a regular expression that captures a value for the $fastcgi_path_info variable. A regular expression should have two captures: the first becomes a value of the $fastcgi_script_name variable, the second becomes a value of the $fastcgi_path_info variable. For example, with these settings

```plaintext
location ~ ^(.+\.php ) (.*) $ {
    fastcgi_split_path_info ^(.+\.php ) (.*) $;
    fastcgi_param SCRIPT_FILENAME /path/to/php$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
}
```

and the “/show.php/article/0001” request, the SCRIPT_FILENAME parameter will be equal to “/path/to/php/show.php”, and the PATH_INFO parameter will be equal to “/article/0001”.

fastcgi_store

**SYNTAX:**  fastcgi_store on | off | string;

**DEFAULT** off

**CONTEXT:** http, server, location

Enables saving of files to a disk. The on parameter saves files with paths corresponding to the directives alias or root. The off parameter disables saving of files. In addition, the file name can be set explicitly using the string with variables:

```plaintext
fastcgi_store /data/www$original_uri;
```

The modification time of files is set according to the received Last-Modified response header field. A response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the persistent store can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both saved files and a directory holding temporary files, set by the fastcgi_temp_path directive, are put on the same file system.

This directive can be used to create local copies of static unchangeable files, e.g.:

```plaintext
location /images/ {
    root /data/www;
    error_page 404 = /fetch$uri;
}
location /fetch/ {
    internal;
    fastcgi_pass backend:9000;
}
```
fastcgi_store_access

SYNTAX:  fastcgi_store_access users:permissions ...;
DEFAULT  user:rw
CONTEXT:  http, server, location

Sets access permissions for newly created files and directories, e.g.:

```
fastcgi_store_access user:rw group:rw all:r;
```

If any group or all access permissions are specified then user permissions may be omitted:

```
fastcgi_store_access group:rw all:r;
```

fastcgi_temp_file_write_size

SYNTAX:  fastcgi_temp_file_write_size size;
DEFAULT  8k|16k
CONTEXT:  http, server, location

Limits the size of data written to a temporary file at a time, when buffering of responses from the FastCGI server to temporary files is enabled. By default, size is limited by two buffers set by the fastcgi_buffer_size and fastcgi_buffers directives. The maximum size of a temporary file is set by the fastcgi_max_temp_file_size directive.

fastcgi_temp_path

SYNTAX:  fastcgi_temp_path path [level1 [level2 [level3]]];
DEFAULT  fastcgi_temp
CONTEXT:  http, server, location

Defines a directory for storing temporary files with data received from FastCGI servers. Up to three-level subdirectory hierarchy can be used underneath the specified directory. For example, in the following configuration

```
fastcgi_temp_path /spool/nginx/fastcgi_temp 1 2;
```

a temporary file might look like this:

```
/spool/nginx/fastcgi_temp/7/45/00000123457
```
2.11.4 Parameters Passed to a FastCGI Server

HTTP request header fields are passed to the FastCGI server as parameters. In applications and scripts running as FastCGI servers, these parameters are usually made available as environment variables. For example, the User-Agent header field is passed as the HTTP_USER_AGENT parameter. In addition to HTTP request header fields, it is possible to pass arbitrary parameters using the fastcgi_param directive.

2.11.5 Embedded Variables

The ngx_http_fastcgi_module module supports embedded variables that can be used to set parameters using the fastcgi_param directive:

```bash
$fastcgi_script_name
```

request URI or, if a URI ends with a slash, request URI with an index file name configured by the fastcgi_index directive appended to it. This variable can be used to set the SCRIPT_FILENAME and PATH_TRANSLATED parameters that determine the script name in PHP. For example, for the “/info/” request with the following directives

```bash
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME /home/www/scripts/ php$fastcgi_script_name;
```

the SCRIPT_FILENAME parameter will be equal to “/home/www/scripts/php/info/index.php”. When using the fastcgi_split_path_info directive, the $fastcgi_script_name variable equals the value of the first capture set by the directive.

```bash
$fastcgi_path_info
```

the value of the second capture set by the fastcgi_split_path_info directive. This variable can be used to set the PATH_INFO parameter.
CHAPTER 2. HTTP SERVER MODULES

2.12 Module ngx_http_f4f_module

2.12.1 Summary

The ngx_http_f4f_module module provides server-side support for Adobe HTTP Dynamic Streaming (HDS).

This module implements handling of HTTP Dynamic Streaming requests in the "/videoSeg1-Frag1" form — extracting the needed fragment from the videoSeg1.f4f file using the videoSeg1.f4x index file. This module is an alternative to the Adobe’s f4f module (HTTP Origin Module) for Apache.

Usual pre-processing with Adobe’s f4fpackager is required, see relevant documentation for details.

This module is available as part of our commercial subscription.

2.12.2 Example Configuration

```plaintext
location /video/ {
  f4f;
  ...
}
```

2.12.3 Directives

**f4f**

SYNTAX: `f4f;`

DEFAULT —

CONTEXT: location

Turns on module processing in the surrounding location.

**f4f_buffer_size**

SYNTAX: `f4f_buffer_size size;`

DEFAULT `512k`

CONTEXT: http, server, location

Sets the size of a memory buffer used for reading the .f4x index file.
2.13 Module ngx_http_flv_module

2.13.1 Summary

The ngx_http_flv_module module provides pseudo-streaming server-side support for Flash Video (FLV) files.

It handles requests with the start argument in the request URI’s query string specially, by sending back the contents of a file starting from the requested byte offset and with the prepended FLV header.

This module is not built by default, it should be enabled with the --with-http_flv_module configuration parameter.

2.13.2 Example Configuration

```
location ~ \.(flv)$ {
  flv;
}
```

2.13.3 Directives

flv

SYNTAX:  flv;

DEFAULT  —

CONTEXT: location

Turns on module processing in a surrounding location.
2.14 Module ngx_http_geoip_module

2.14.1 Summary

The `ngx_http_geoip_module` module (0.8.6+) creates variables with values depending on the client IP address, using the precompiled MaxMind databases.

When using the databases with IPv6 support (1.3.12, 1.2.7), IPv4 addresses are looked up as IPv4-mapped IPv6 addresses.

This module is not built by default, it should be enabled with the `--with-http_geoip_module` configuration parameter.

This module requires the MaxMind GeoIP library.

2.14.2 Example Configuration

```nginx
http {
    geoip_country GeoIP.dat;
    geoip_city GeoLiteCity.dat;
    geoip_proxy 192.168.100.0/24;
    geoip_proxy 2001:0:db8::/32;
    geoip_proxy_recursive on;
    ...
}
```

2.14.3 Directives

**geoip_country**

SYNTAX:  `geoip_country file;`
DEFAULT —
CONTEXT: http

Specifies a database used to determine the country depending on the client IP address. The following variables are available when using this database:

- `$geoip_country_code`
  two-letter country code, for example, “RU”, “US”.
- `$geoip_country_code3`
  three-letter country code, for example, “RUS”, “USA”.
- `$geoip_country_name`
  country name, for example, “Russian Federation”, “United States”.

**geoip_city**

SYNTAX:  `geoip_city file;`
DEFAULT —
CONTEXT: http
Specifies a database used to determine the country, region, and city depending on the client IP address. The following variables are available when using this database:

$geoip_area_code
telephone area code (US only).

This variable may contain outdated information since the corresponding database field is deprecated.

$geoip_city_continent_code
two-letter continent code, for example, “EU”, “NA”.

$geoip_city_country_code
two-letter country code, for example, “RU”, “US”.

$geoip_city_country_code3
three-letter country code, for example, “RUS”, “USA”.

$geoip_city_country_name
country name, for example, “Russian Federation”, “United States”.

$geoip_dma_code
DMA region code in US (also known as “metro code”), according to the geotargeting in Google AdWords API.

$geoip_latitude
latitude.

$geoip_longitude
longitude.

$geoip_region
two-symbol country region code (region, territory, state, province, federal land and the like), for example, “48”, “DC”.

$geoip_region_name
country region name (region, territory, state, province, federal land and the like), for example, “Moscow City”, “District of Columbia”.

$geoip_city
city name, for example, “Moscow”, “Washington”.

$geoip_postal_code
postal code.

**geoip_org**

SYNTAX: `geoip_org file;`

DEFAULT —

CONTEXT: http

This directive appeared in version 1.0.3.

Specifies a database used to determine the organization depending on the client IP address. The following variable is available when using this database:

$geoip_org
organization name, for example, “The University of Melbourne”.

Nginx, Inc.  p.96 of 242
geoip_proxy

SYNTAX: geoip_proxy address | CIDR;
DEFAULT —
CONTEXT: http
This directive appeared in versions 1.3.0 and 1.2.1.

Defines trusted addresses. When a request comes from a trusted address, an address from the X-Forwarded-For request header field will be used instead.

geoip_proxy_recursive

SYNTAX: geoip_proxy_recursive on | off;
DEFAULT off
CONTEXT: http
This directive appeared in versions 1.3.0 and 1.2.1.

If recursive search is disabled then instead of the original client address that matches one of the trusted addresses, the last address sent in X-Forwarded-For will be used. If recursive search is enabled then instead of the original client address that matches one of the trusted addresses, the last non-trusted address sent in X-Forwarded-For will be used.
2.15 Module ngx_http_geo_module

2.15.1 Summary

The ngx_http_geo_module module creates variables with values depending on the client IP address.

2.15.2 Example Configuration

```
geo $geo {
  default 0;
  127.0.0.1 2;
  192.168.1.0/24 1;
  10.1.0.0/16 1;
  ::1 2;
  2001:0 db8 ::/32 1;
}
```

2.15.3 Directives

```
geo $geo {
  default 0;
  127.0.0.1 2;
  192.168.1.0/24 1;
  10.1.0.0/16 1;
  ::1 2;
  2001:0 db8 ::/32 1;
}
```

Describes the dependency of values of the specified variable on the client IP address. By default, the address is taken from the $remote_addr variable, but it can also be taken from another variable (0.7.27), for example:

```
geo $arg_remote_addr $geo {
  ...
}
```

Since variables are evaluated only when used, the mere existence of even a large number of declared “geo” variables does not cause any extra costs for request processing.

If the value of a variable does not represent a valid IP address then the “255.255.255.255” address is used.

Addresses are specified either as prefixes in CIDR notation (including individual addresses) or as ranges (0.7.23).

IPv6 prefixes are supported starting from versions 1.3.10 and 1.2.7.

The following special parameters are also supported:
delete
 deletes the specified network (0.7.23).

default
 a value set to the variable if the client address does not match any of
 the specified addresses. When addresses are specified in CIDR notation,
 “0.0.0.0/0” and “::/0” can be used instead of default. When default
 is not specified, the default value will be an empty string.

include
 includes a file with addresses and values. There can be several inclusions.

proxy
 defines trusted addresses (0.8.7, 0.7.63). When a request comes from a
 trusted address, an address from the X-Forwarded-For request header
 field will be used instead. In contrast to the regular addresses, trusted
 addresses are checked sequentially.

proxy_recursive
 enables recursive address search (1.3.0, 1.2.1). If recursive search is
 disabled then instead of the original client address that matches one
 of the trusted addresses, the last address sent in X-Forwarded-For will
 be used. If recursive search is enabled then instead of the original client
 address that matches one of the trusted addresses, the last non-trusted
 address sent in X-Forwarded-For will be used.

ranges
 indicates that addresses are specified as ranges (0.7.23). This parameter
 should be the first. To speed up loading of a geo base, addresses should
 be put in ascending order.

Example:

```bash
geo $country {
  default ZZ;
  include conf/geo.conf;
  delete 127.0.0.0/16;
  proxy 192.168.100.0/24;
  proxy_recursive
  127.0.0.0/24 US;
  127.0.0.1/32 RU;
  10.1.0.0/16 RU;
  192.168.1.0/24 UK;
}
```

The `conf/geo.conf` file could contain the following lines:

```bash
10.2.0.0/16 RU;
192.168.2.0/24 RU;
```

A value of the most specific match is used. For example, for the 127.0.0.1
address the value “RU” will be chosen, not “US”.

Nginx, Inc.  p.99 of 242
Example with ranges:

```plaintext
geo $country {
  ranges;
  default ZZ;
  127.0.0.0 -127.0.0.0 US;
  127.0.0.1 -127.0.0.1 RU;
  127.0.0.1 -127.0.0.255 US;
  10.1.0.0 -10.1.255.255 RU;
  192.168.1.0 -192.168.1.255 UK;
}
```
2.16 Module ngx_http_gunzip_module

2.16.1 Summary

The ngx_http_gunzip_module module is a filter that decompresses responses with “Content-Encoding: gzip” for clients that do not support “gzip” encoding method. The module will be useful when it is desirable to store data compressed to save space and reduce I/O costs.

This module is not built by default, it should be enabled with the --with-http_gunzip_module configuration parameter.

2.16.2 Example Configuration

```
location /storage/ {
    gunzip on;
    ...
}
```

2.16.3 Directives

gunzip

**SYNTAX:** gunzip on | off;
**DEFAULT** off
**CONTEXT:** http, server, location

Enables or disables decompression of gzipped responses for clients that lack gzip support. If enabled, the following directives are also taken into account when determining if clients support gzip: gzip_http_version, gzip_proxied, and gzip_disable. See also the gzip_vary directive.

gunzip_buffers

**SYNTAX:** gunzip_buffers number size;
**DEFAULT** 32 4k|16 8k
**CONTEXT:** http, server, location

Sets the number and size of buffers used to decompress a response. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.
2.17 Module ngx_http_gzip_module

2.17.1 Summary

The ngx_http_gzip_module module is a filter that compresses responses using the “gzip” method. This often helps to reduce the size of transmitted data by half or even more.

2.17.2 Example Configuration

```
gzip on;
gzip_min_length 1000;
gzip_proxied expired no-cache no-store private auth;
gzip_types text/plain application/xml;
```

The $gzip_ratio variable can be used to log the achieved compression ratio.

2.17.3 Directives

**gzip**

- **SYNTAX:** gzip on | off;
- **DEFAULT:** off
- **CONTEXT:** http, server, location, if in location

Enables or disables gzipping of responses.

**gzip_buffers**

- **SYNTAX:** gzip_buffers number size;
- **DEFAULT:** 32 4k|16 8k
- **CONTEXT:** http, server, location

Sets the number and size of buffers used to compress a response. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

Until version 0.7.28, four 4K or 8K buffers were used by default.

**gzip_comp_level**

- **SYNTAX:** gzip_comp_level level;
- **DEFAULT:** 1
- **CONTEXT:** http, server, location

Sets a gzip compression level of a response. Acceptable values are in the range from 1 to 9.
gzip_disable

SYNTAX: gzip_disable regex ...;
DEFAULT —
CONTEXT: http, server, location
This directive appeared in version 0.6.23.

Disables gzipping of responses for requests with User-Agent header fields matching any of the specified regular expressions.

The special mask “msie6” (0.7.12) corresponds to the regular expression “MSIE [4-6] \.”, but works faster. Starting from version 0.8.11, “MSIE 6.0; ...SV1” is excluded from this mask.

gzip_min_length

SYNTAX: gzip_min_length length;
DEFAULT 20
CONTEXT: http, server, location

Sets the minimum length of a response that will be gzipped. The length is determined only from the Content-Length response header field.

gzip_http_version

SYNTAX: gzip_http_version 1.0 | 1.1;
DEFAULT 1.1
CONTEXT: http, server, location

Sets the minimum HTTP version of a request required to compress a response.

gzip_proxied

SYNTAX: gzip_proxied off | expired | no-cache | no-store | private |
	| no_last_modified | no_etag | auth | any ...;
DEFAULT off
CONTEXT: http, server, location

Enables or disables gzipping of responses for proxied requests depending on the request and response. The fact that the response is proxied is determined by the presence of the Via request header field. A directive accepts multiple parameters:

off

disables compression for all proxied requests, ignoring other parameters;
expired

enables compression if a response header includes the Expires field with a value that disables caching;
nocache

enables compression if a response header includes the Cache-Control field with the “no-cache” parameter;
no-store
  enables compression if a response header includes the Cache-Control field
  with the “no-store” parameter;

private
  enables compression if a response header includes the Cache-Control field
  with the “private” parameter;

no_last_modified
  enables compression if a response header does not include the Last-Modified field;

no_etag
  enables compression if a response header does not include the ETag field;

auth
  enables compression if a request header includes the Authorization field;

any
  enables compression for all proxied requests.

gzip_types
  
  SYNTAX:  gzip_types mime-type ...;
  DEFAULT text/html
  CONTEXT: http, server, location

  Enables gzipping of responses for the specified MIME types in addition to “text/html”. The special value “*” matches any MIME type (0.8.29). Responses with the “text/html” type are always compressed.

gzip_vary
  
  SYNTAX:  gzip_vary on | off;
  DEFAULT off
  CONTEXT: http, server, location

  Enables or disables inserting the Vary: Accept-Encoding response header field if the directives gzip, gzip_static, or gunzip are active.

2.17.4 Embedded Variables

$gzip_ratio
  achieved compression ratio, computed as the ratio between the original and compressed response sizes.
2.18 Module ngx_http_gzip_static_module

2.18.1 Summary

The ngx_http_gzip_static_module module allows sending precompressed files with the “.gz” filename extension instead of regular files.

This module is not built by default, it should be enabled with the --with-http_gzip_static_module configuration parameter.

2.18.2 Example Configuration

```
gzip_static on;
gzip_proxied expired no-cache no-store private auth;
```

2.18.3 Directives

**gzip_static**

SYNTAX:  gzip_static on | off | always;

DEFAULT:  off

CONTEXT:  http, server, location

Enables (“on”) or disables (“off”) checking the existence of precompressed files. The following directives are also taken into account: gzip_http_version, gzip_proxied, gzip_disable, and gzip_vary.

With the “always” value (1.3.6), gzipped file is used in all cases, without checking if the client supports it. It is useful if there are no uncompressed files on the disk anyway or the ngx_http_gunzip_module is used.

The files can be compressed using the gzip command, or any other compatible one. It is recommended that the modification date and time of original and compressed files be the same.
2.19 Module ngx_http_headers_module

2.19.1 Summary

The ngx_http_headers_module module allows adding the Expires and Cache-Control header fields, and arbitrary fields, to a response header.

2.19.2 Example Configuration

<table>
<thead>
<tr>
<th>Directive</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>expires</td>
<td>24h;</td>
</tr>
<tr>
<td>expires</td>
<td>modified +24h;</td>
</tr>
<tr>
<td>expires</td>
<td>0;</td>
</tr>
<tr>
<td>expires</td>
<td>-1;</td>
</tr>
<tr>
<td>expires</td>
<td>epoch;</td>
</tr>
<tr>
<td>add_header</td>
<td>Cache-Control private;</td>
</tr>
</tbody>
</table>

2.19.3 Directives

add_header

**SYNTAX:** add_header name value;

**DEFAULT:** —

**CONTEXT:** http, server, location, if in location

Adds the specified field to a response header provided that the response code equals 200, 201, 204, 206, 301, 302, 303, 304, or 307. A value can contain variables.

There could be several add_header directives. These directives are inherited from the previous level if and only if there are no add_header directives defined on the current level.

expires

**SYNTAX:** expires [modified] time;

**SYNTAX:** expires epoch | max | off;

**DEFAULT:** off

**CONTEXT:** http, server, location, if in location

Enables or disables adding or modifying the Expires and Cache-Control response header fields provided that the response code equals 200, 201, 204, 206, 301, 302, 303, 304, or 307. A parameter can be a positive or negative time.

A time in the Expires field is computed as a sum of the current time and time specified in the directive. If the modified parameter is used (0.7.0, 0.6.32) then time is computed as a sum of the file’s modification time and time specified in the directive.

In addition, it is possible to specify a time of the day using the “@” prefix (0.7.9, 0.6.34):
The epoch parameter corresponds to the absolute time “Thu, 01 Jan 1970 00:00:01 GMT”. The contents of the Cache-Control field depends on the sign of the specified time:

- time is negative — Cache-Control: no-cache.
- time is positive or zero — Cache-Control: max-age=t, where t is a time specified in the directive, in seconds.

The max parameter sets Expires to the value “Thu, 31 Dec 2037 23:55:55 GMT”, and Cache-Control to 10 years.

The off parameter disables adding or modifying the Expires and Cache-Control response header fields.
2.20 Module ngx_http_hls_module

2.20.1 Summary

The ngx_http_hls_module module provides HTTP Live Streaming (HLS) server-side support for H.264/AAC files. Such files typically have the .mp4, .m4v, or .m4a filename extensions. nginx supports two URIs for each MP4 file:

- The playlist URI that ends with ".m3u8" and accepts the optional "len" argument that defines the fragment length in seconds;
- The fragment URI that ends with ".ts" and accepts "start" and "end" arguments that define fragment boundaries in seconds.

This module is available as part of our commercial subscription.

2.20.2 Example Configuration

```configuration
location /video/ {
  hls;
  hls_fragment 5s;
  hls_buffers 10 10m;
  hls_mp4_buffer_size 1m;
  hls_mp4_max_buffer_size 5m;
  alias /var/video/;
}
```

With this configuration, the following URIs are supported for the "/var/video/test.mp4" file:

- `http://hls.example.com/video/test.mp4.m3u8?len=8.000`

2.20.3 Directives

**hls**

**SYNTAX:** hls;

**DEFAULT:** —

**CONTEXT:** location

Turns on HLS streaming in the surrounding location.

**hls_buffers**

**SYNTAX:** hls_buffers number size;

**DEFAULT:** 8 2m

**CONTEXT:** http, server, location

Sets the maximum `number` and `size` of buffers that are used for reading and writing data frames.
hls_fragment

SYNTAX: `hls_fragment time;`

DEFAULT: 5s

CONTEXT: http, server, location

Defines the default fragment length for playlist URIs requested without the "len" argument.

hls_mp4_buffer_size

SYNTAX: `hls_mp4_buffer_size size;`

DEFAULT: 512k

CONTEXT: http, server, location

Sets the initial size of the memory buffer used to process MP4 files.

hls_mp4_max_buffer_size

SYNTAX: `hls_mp4_max_buffer_size size;`

DEFAULT: 10m

CONTEXT: http, server, location

During metadata processing, a larger buffer may become necessary. Its size cannot exceed the specified size, or else nginx will return the server error 500 Internal Server Error, and log the following message:

"/some/movie/file.mp4" mp4 moov atom is too large:
12583268, you may want to increase hls_mp4_max_buffer_size
2.21 Module ngx_http_image_filter_module

2.21.1 Summary

The ngx_http_image_filter_module module (0.7.54+) is a filter that transforms images in JPEG, GIF, and PNG formats. This module is not built by default, it should be enabled with the --with-http_image_filter_module configuration parameter.

This module utilizes the libgd library. It is recommended to use the latest available version of the library.

2.21.2 Example Configuration

```nginx
location /img/ {
    proxy_pass http://backend;
    image_filter resize 150 100;
    image_filter rotate 90;
    error_page 415 = /empty;
}

location = /empty {
    empty.gif;
}
```

2.21.3 Directives

- **image_filter**
  - **syntax:** image_filter off;
  - **syntax:** image_filter test;
  - **syntax:** image_filter size;
  - **syntax:** image_filter rotate 90 | 180 | 270;
  - **syntax:** image_filter resize width height;
  - **syntax:** image_filter crop width height;
  - **default:** off
  - **context:** location

Sets the type of transformation to perform on images:

- **off**
  - turns off module processing in a surrounding location.
- **test**
  - ensures that responses are images in either JPEG, GIF, or PNG format. Otherwise, the 415 Unsupported Media Type error is returned.
- **size**
  - outputs information about images in a JSON format, e.g.:

```json
{ "img" : { "width": 100, "height": 100, "type": "gif" } }
```
In case of an error, the output is as follows:

```
{}
```

**rotate 90|180|270**

rotates images counter-clockwise by the specified number of degrees. Parameter value can contain variables. This mode can be used either alone or along with the **resize** and **crop** transformations.

**resize width height**

proportionally reduces an image to the specified sizes. To reduce by only one dimension, another dimension can be specified as “-”. In case of an error, the server will return code **415 Unsupported Media Type**. Parameter values can contain variables. When used along with the **rotate** parameter, the rotation happens after reduction.

**crop width height**

proportionally reduces an image to the larger side size and crops extraneous edges by another side. To reduce by only one dimension, another dimension can be specified as “-”. In case of an error, the server will return code **415 Unsupported Media Type**. Parameter values can contain variables. When used along with the **rotate** parameter, the rotation happens before reduction.

**image_filter_buffer**

**SYNTAX:** `image_filter_buffer size;`
**DEFAULT:** 1M
**CONTEXT:** http, server, location

Sets the maximum size of the buffer used for reading images. When the size is exceeded the server returns error **415 Unsupported Media Type**.

**image_filter_interlace**

**SYNTAX:** `image_filter_interlace on | off;`
**DEFAULT:** off
**CONTEXT:** http, server, location

This directive appeared in version 1.3.15.

If enabled, final images will be interlaced. For JPEG, final images will be in “progressive JPEG” format.

**image_filter_jpeg_quality**

**SYNTAX:** `image_filter_jpeg_quality quality;`
**DEFAULT:** 75
**CONTEXT:** http, server, location

Sets the desired *quality* of the transformed JPEG images. Acceptable values are in the range from 1 to 100. Lesser values usually imply both lower image
quality and less data to transfer. The maximum recommended value is 95. Parameter value can contain variables.

image_filter_sharpen

SYNTAX:  image_filter_sharpen percent;
DEFAULT 0
CONTEXT: http, server, location

Increases sharpness of the final image. The sharpness percentage can exceed 100. The zero value disables sharpening. Parameter value can contain variables.

image_filter_transparency

SYNTAX:  image_filter_transparency on|off;
DEFAULT on
CONTEXT: http, server, location

Defines whether transparency should be preserved when transforming GIF images or PNG images with colors specified by a palette. The loss of transparency results in images of a better quality. The alpha channel transparency in PNG is always preserved.
## 2.22 Module ngx_http_index_module

### 2.22.1 Summary

The `ngx_http_index_module` module processes requests ending with the slash character (`/`). Such requests can also be processed by the `ngx_http-autoindex_module` and `ngx_http_random_index_module` modules.

### 2.22.2 Example Configuration

```bash
location / {
    index index.$geo.html index.html;
}
```

### 2.22.3 Directives

#### index

**Syntax:** `index file ...;`

**Default:** `index.html`

**Context:** `http, server, location`

Defines files that will be used as an index. The `file` name can contain variables. Files are checked in the specified order. The last element of the list can be a file with an absolute path. Example:

```bash
index index.$geo.html index.0.html /index.html;
```

It should be noted that using an index file causes an internal redirect, and the request can be processed in a different location. For example, with the following configuration:

```bash
location = / {
    index index.html;
}
location / {
    ...
}
```

a “/” request will actually be processed in the second location as “/index.html”.

Nginx, Inc.  p.113 of 242
2.23 Module ngx_http_limit_conn_module

2.23.1 Summary

The ngx_http_limit_conn_module module is used to limit the number of connections per the defined key, in particular, the number of connections from a single IP address.

Not all connections are counted. A connection is counted only if it has a request processed by the server and the whole request header has already been read.

2.23.2 Example Configuration

```plaintext
http {
    limit_conn_zone $binary_remote_addr zone=addr :10m;
    ... 
    server {
        ... 
        location /download/ {
            limit_conn addr 1;
        }
    }
}
```

2.23.3 Directives

**limit_conn**

SYNTAX: `limit_conn zone number;`

DEFAULT 

CONTEXT: http, server, location

Sets the shared memory zone and the maximum allowed number of connections for a given key value. When this limit is exceeded, the server will return the `503 Service Temporarily Unavailable` error in reply to a request. For example, the directives

```plaintext
limit_conn_zone $binary_remote_addr zone=addr :10m;
server {
    location /download/ {
        limit_conn addr 1;
    }
}
```

allow only one connection per an IP address at a time.

When several `limit_conn` directives are specified, any configured limit will apply. For example, the following configuration will limit the number of connections to the server per a client IP and, at the same time, the total number of connections to the virtual host:
These directives are inherited from the previous level if and only if there are no `limit_conn` directives on the current level.

### limit_conn_log_level

**SYNTAX:** `limit_conn_log_level info | notice | warn | error;`

**DEFAULT:** `error`

**CONTEXT:** `http, server, location`

This directive appeared in version 0.8.18.

Sets the desired logging level for cases when the server limits the number of connections.

### limit_conn_status

**SYNTAX:** `limit_conn_status code;`

**DEFAULT:** `503`

**CONTEXT:** `http, server, location`

This directive appeared in version 1.3.15.

Sets the status code to return in response to rejected requests.

### limit_conn_zone

**SYNTAX:** `limit_conn_zone $variable zone=name:size;`

**DEFAULT:** `—`

**CONTEXT:** `http`

Sets parameters for a shared memory zone that will keep states for various keys. In particular, the state includes the current number of connections. The key is any non-empty value of the specified variable (empty values are not accounted). Usage example:

```
limit_conn_zone $binary_remote_addr zone=addr:10m;
```

Here, a client IP address serves as a key. Note that instead of `$remote_addr`, the `$binary_remote_addr` variable is used here. The `$remote_addr` variable’s size can vary from 7 to 15 bytes. The stored state occupies either 32 or 64 bytes of memory on 32-bit platforms and always 64 bytes on 64-bit platforms. The `$binary_remote_addr` variable’s size is always 4 bytes. The stored state always occupies 32 bytes on 32-bit platforms and 64 bytes on 64-bit platforms. One megabyte zone can keep about 32 thousand 32-byte states or about 16...
thousand 64-byte states. If the zone storage is exhausted, the server will return the 503 Service Temporarily Unavailable error to all further requests.

**limit_zone**

**SYNTAX:**  `limit_zone name $variable size;`

**DEFAULT**  —

**CONTEXT:**  `http`

This directive is made obsolete in version 1.1.8, an equivalent `limit_conn_zone` directive with a changed syntax should be used instead:

```
limit_conn_zone $variable zone=name:size;
```
2.24 Module ngx_http_limit_req_module

2.24.1 Summary

The ngx_http_limit_req_module module (0.7.21) is used to limit the request processing rate per a defined key, in particular, the processing rate of requests coming from a single IP address. The limitation is done using the “leaky bucket” method.

2.24.2 Example Configuration

```plaintext
http {
  limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
  ...
  server {
    ...
    location /search/ {
      limit_req zone=one burst=5;
    }
  }
}
```

2.24.3 Directives

**limit_req**

SYNTAX:  `limit_req zone=name [burst=number] [nodelay];`

DEFAULT —

CONTEXT:  http, server, location

Sets the shared memory zone and the maximum burst size of requests. If the requests rate exceeds the rate configured for a zone, their processing is delayed such that requests are processed at a defined rate. Excessive requests are delayed until their number exceeds the maximum burst size in which case the request is terminated with an error 503 Service Temporarily Unavailable. By default, the maximum burst size is equal to zero. For example, the directives

```plaintext
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
server {
  location /search/ {
    limit_req zone=one burst=5;
  }
}
```

allow not more than 1 request per second at an average, with bursts not exceeding 5 requests.

If delaying of excessive requests while requests are being limited is not desired, the parameter nodelay should be used:
limit_req zone=one burst=5 nodelay;

**limit req log level**

**Syntax:** limit req log level info | notice | warn | error;

**Default:** error

**Context:** http, server, location

This directive appeared in version 0.8.18.

Sets the desired logging level for cases when the server refuses to process requests due to rate exceeding, or delays request processing. Logging level for delays is one point less than for refusals; for example, if “limit req log level notice” is specified, delays are logged with the info level.

**limit req status**

**Syntax:** limit req status code;

**Default:** 503

**Context:** http, server, location

This directive appeared in version 1.3.15.

Sets the status code to return in response to rejected requests.

**limit req zone**

**Syntax:** limit req zone $variable zone=name:size rate=rate;

**Default:** —

**Context:** http

Sets parameters for a shared memory zone that will keep states for various keys. In particular, the state stores the current number of excessive requests. The key is any non-empty value of the specified variable (empty values are not accounted). Usage example:

```
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
```

Here, the states are kept in a 10 megabyte zone “one”, and an average request processing rate for this zone cannot exceed 1 request per second.

A client IP address serves as a key. Note that instead of $remote_addr, the $binary_remote_addr variable is used here, that allows to decrease the state size down to 64 bytes. One megabyte zone can keep about 16 thousand 64-byte states. If the zone storage is exhausted, the server will return the 503 Service Temporarily Unavailable error to all further requests.

The rate is specified in requests per second (r/s). If a rate of less than one request per second is desired, it is specified in request per minute (r/m). For example, half-request per second is 30r/m.
Module ngx_http_log_module

2.25.1 Summary

The ngx_http_log_module module writes request logs in the specified format.

Requests are logged in the context of a location where processing ends. It may be different from the original location, if an internal redirect happens during request processing.

Example Configuration

```
log_format compression '$_remote_addr - $remote_user [$_time_local] ' 
'$_request $status $bytes_sent ' 
'$_http_referrer ''$_http_user_agent'' $gzip_ratio''

access_log /spool/logs/nginx-access.log compression buffer=32k;
```

2.25.3 Directives

*access_log*

**SYNTAX:**  access_log path [format [buffer=size [flush=time]]];

**SYNTAX:**  access_log path format gzip=[level] [buffer=size] [flush=time];

**SYNTAX:**  access_log syslog:server=address[parameter=value] [format];

**DEFAULT**  access_log off;

**CONTEXT:** http, server, location, if in location, limit, except

Sets the path, format, and configuration for a buffered log write. Several logs can be specified on the same level. Logging to syslog can be configured by specifying the “syslog:” prefix in the first parameter. The special value off cancels all access_log directives on the current level. If the format is not specified then the predefined “combined” format is used.

If either the buffer or gzip (1.3.10, 1.2.7) parameter is used, writes to log will be buffered.

The buffer size must not exceed the size of an atomic write to a disk file. For FreeBSD this size is unlimited.

When buffering is enabled, the data will be written to the file:

- if the next log line does not fit into the buffer;
- if the buffered data is older than specified by the flush parameter (1.3.10, 1.2.7);
- when a worker process is re-opening log files or is shutting down.
If the gzip parameter is used, then the buffered data will be compressed before writing to the file. The compression level can be set between 1 (fastest, less compression) and 9 (slowest, best compression). By default, the buffer size is equal to 64K bytes, and the compression level is set to 1. Since the data is compressed in atomic blocks, the log file can be decompressed or read by “zcat” at any time.

Example:

```nginx
access_log /path/to/log.gz combined gzip flush=5m;
```

For gzip compression to work, nginx must be built with the zlib library.

The file path can contain variables (0.7.6+), but such logs have some constraints:

- the user whose credentials are used by worker processes should have permissions to create files in a directory with such logs;
- buffered writes do not work;
- the file is opened and closed for each log write. However, since the descriptors of frequently used files can be stored in a cache, writing to the old file can continue during the time specified by the `open_log_file_cache` directive’s `valid` parameter
- during each log write the existence of the request’s root directory is checked, and if it does not exist the log is not created. It is thus a good idea to specify both root and `access_log` on the same level:

```nginx
server {
    root /spool/vhost/data/$host;
    access_log /spool/vhost/logs/$host;
    ...
}
```

The following parameters configure logging to syslog:

**server=** *address*

Defines the address of a syslog server. The address can be specified as a domain name, IP address, or a UNIX-domain socket path (specified after the “unix:” prefix). With a domain name or IP address, the port can be specified. If port is not specified, the port 514 is used. If a domain name resolves to several IP addresses, the first resolved address is used.

**facility=** *string*

Sets facility of syslog messages, as defined in RFC 3164. Facility can be one of “kern”, “user”, “mail”, “daemon”, “auth”, “intern”, “lpr”, “news”, “uucp”, “clock”, “authpriv”, “ftp”, “ntp”, “audit”, “alert”, “cron”, “local0”..”local7”. Default is “local7”.

Nginx, Inc. p.120 of 242
severity=string
Sets severity of syslog messages, as defined in RFC 3164. Possible values are the same as for the second parameter (level) of the error_log directive. Default is "info".

tag=string
Sets the tag of syslog messages. Default is "nginx".

Example syslog configuration:

```
access_log syslog:server=192.168.1.1;
access_log syslog:server=unix:/var/log/nginx.sock;
access_log syslog:server=[2001:db8::1]:12345,facility=local7,tag=nginx,
    severity=info,combined;
```

Logging to syslog is available as part of our commercial subscription.

log_format

SYNTAX: log_format name string...;
DEFAULT combined "...

CONTEXT: http

Specifies log format.
The log format can contain common variables, and variables that exist only at the time of a log write:

$bytes_sent
the number of bytes sent to a client

$connection
connection serial number

$connection_requests
the current number of requests made through a connection (1.1.18)

$msec
time in seconds with a milliseconds resolution at the time of the log write

$pipe
"p" if request was pipelined, "." otherwise

$request_length
request length (including request line, header, and request body)

$request_time
request processing time in seconds with a milliseconds resolution; time elapsed between the first bytes were read from the client and the log write after the last bytes were sent to the client

$status
response status

$time_iso8601
local time in the ISO 8601 standard format
$\text{time\_local}$
local time in the Common Log Format

In the modern nginx versions variables $\text{status}$ (1.3.2, 1.2.2), $\text{bytes\_sent}$ (1.3.8, 1.2.5), $\text{connection}$ (1.3.8, 1.2.5), $\text{connection\_requests}$ (1.3.8, 1.2.5), $\text{msec}$ (1.3.9, 1.2.6), $\text{request\_time}$ (1.3.9, 1.2.6), $\text{pipe}$ (1.3.12, 1.2.7), $\text{request\_length}$ (1.3.12, 1.2.7), $\text{time\_iso8601}$ (1.3.12, 1.2.7), and $\text{time\_local}$ (1.3.12, 1.2.7) are also available as common variables.

Header lines sent to a client have the prefix “sent\_http\_”, for example, $\text{sent\_http\_content\_range}$.

The configuration always includes the predefined “combined” format:

```
log_format combined ' $remote_addr - $remote_user \[ $time_local \] '                       
'' $request "$status $body_bytes_sent '          
'' $http_referer "$http_user_agent" ';
```

**open\_log\_file\_cache**

**SYNTAX:**

```
open_log_file_cache max=N [inactive=time] [min_uses=N]  
[valid=time];
```

**DEFAULT**

`off`

**CONTEXT:** http, server, location

Defines a cache that stores the file descriptors of frequently used logs whose names contain variables. The directive has the following parameters:

**max**
sets the maximum number of descriptors in a cache; if the cache becomes full the least recently used (LRU) descriptors are closed

**inactive**
sets the time after which the cached descriptor is closed if there were no access during this time; by default, 10 seconds

**min\_uses**
sets the minimum number of file uses during the time defined by the `inactive` parameter to let the descriptor stay open in a cache; by default, 1

**valid**
sets the time after which it should be checked that the file still exists with the same name; by default, 60 seconds

**off**

disables caching

Usage example:

```
open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;
```
2.26 Module ngx_http_map_module

2.26.1 Summary

The ngx_http_map_module module creates variables whose values depend on values of other variables.

2.26.2 Example Configuration

```plaintext
map $http_host $name {
    hostnames;
    default 0;
    example.com 1;
    *.example.com 1;
    example.org 2;
    *.example.org 2;
    .example.net 3;
    wap.* 4;
}
map $http_user_agent $mobile {
    default 0;
    "Opera Mini" 1;
}
```

2.26.3 Directives

map

**SYNTAX:**  
```plaintext
map string $variable { ... }
```

**DEFAULT** —

**CONTEXT:**  
```plaintext
http
```

Creates a new variable whose value depends on values of one or more of the source variables specified in the first parameter.

Before version 0.9.0 only a single variable could be specified in the first parameter.

Since variables are evaluated only when they are used, the mere declaration even of a large number of "map" variables does not add any extra costs to request processing.

Parameters inside the map block specify a mapping between source and resulting values.

Source values are specified as strings or regular expressions (0.9.6).

A regular expression should either start from the "~" symbol for a case-sensitive matching, or from the "~*" symbols (1.0.4) for case-insensitive matching. A regular expression can contain named and positional captures that can later be used in other directives along with the resulting variable.
If a source value matches one of the names of special parameters described below, it should be prefixed with the "\" symbol.

The resulting value can be a string or another variable (0.9.0).

The directive also supports three special parameters:

**default value**

sets the resulting value if the source value matches none of the specified variants. When `default` is not specified, the default resulting value will be an empty string.

**hostnames**

indicates that source values can be hostnames with a prefix or suffix mask:

```
*.example.com 1;
exanel.*  1;
```

The following two records

```
example.com 1;
*.example.com 1;
```

can be combined:

```
.example.com 1;
```

This parameter should be specified before the list of values.

**include file**

includes a file with values. There can be several inclusions.

If the source value matches more than one of the specified variants, e.g. both a mask and a regular expression match, the first matching variant will be chosen, in the following order of priority:

1. string value without a mask
2. longest string value with a prefix mask, e.g. “*.example.com”
3. longest string value with a suffix mask, e.g. “mail.*”
4. first matching regular expression (in order of appearance in a configuration file)
5. default value

**map_hash_bucket_size**

**SYNTAX:** `map_hash_bucket_size size;`

**DEFAULT:** `32|64|128`

**CONTEXT:** `http`

Sets the bucket size for the map variables hash tables. Default value depends on the processor’s cache line size. The details of setting up hash tables are provided in a separate document.
map_hash_max_size

SYNTAX: map_hash_max_size size;
DEFAULT 2048
CONTEXT: http

Sets the maximum size of the map variables hash tables. The details of setting up hash tables are provided in a separate document.
2.27 Module ngx_http_memcached_module

2.27.1 Summary

The ngx_http_memcached_module module is used to obtain responses from a memcached server. The key is set in the $memcached_key variable. A response should be put in memcached in advance by means external to nginx.

2.27.2 Example Configuration

```nginx
server {
  location / {
    set $memcached_key "$uri?$args";
    memcached_pass host:11211;
    error_page 404 502 504 = @fallback;
  }
  location @fallback {
    proxy_pass http://backend;
  }
}
```

2.27.3 Directives

**memcached_bind**

**SYNTAX:** `memcached_bind address | off;`

**DEFAULT** —

**CONTEXT:** http, server, location

This directive appeared in version 0.8.22.

Makes outgoing connections to a memcached server originate from the specified local IP address. Parameter value can contain variables (1.3.12). The special value `off` (1.3.12) cancels the effect of the `memcached_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address.

**memcached_buffer_size**

**SYNTAX:** `memcached_buffer_size size;`

**DEFAULT** 4k|8k

**CONTEXT:** http, server, location

Sets the size of the buffer used for reading a response received from the memcached server. A response is passed to a client synchronously, as soon as it is received.

**memcached_connect_timeout**

**SYNTAX:** `memcached_connect_timeout time;`

**DEFAULT** 60s

**CONTEXT:** http, server, location
Defines a timeout for establishing a connection with a memcached server. It should be noted that this timeout cannot usually exceed 75 seconds.

### memcached_gzip_flag

**SYNTAX:** `memcached_gzip_flag flag;`

**DEFAULT:**

**CONTEXT:** `http`, `server`, `location`

This directive appeared in version 1.3.6.

Enables the test for the `flag` presence in the memcached server response and sets the “Content-Encoding” response header field to “gzip” if the flag is set.

### memcached_next_upstream

**SYNTAX:** `memcached_next_upstream error | timeout | invalid_response | not_found | off ...;`

**DEFAULT:** `error timeout`

**CONTEXT:** `http`, `server`, `location`

Specifies in which cases a request should be passed to the next server:

- **error**
  - an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

- **timeout**
  - a timeout has occurred while establishing a connection with the server, passing a request to it, or reading the response header;

- **invalid_response**
  - a server returned an empty or invalid response;

- **not_found**
  - a response was not found on the server;

- **off**
  - disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an unsuccessful attempt of communication with a server. The cases of **error**, **timeout** and **invalid_header** are always considered unsuccessful attempts, even if they are not specified in the directive. The case of **not_found** is never considered an unsuccessful attempt.

### memcached_pass

**SYNTAX:** `memcached_pass address;`

**DEFAULT:**

**CONTEXT:** `location`, if in `location`

---

Nginx, Inc.  

p.127 of 242
Sets the memcached server address. The address can be specified as a domain name or an address, and a port:

```
memcached_pass localhost:11211;
```

or as a UNIX-domain socket path:

```
memcached_pass unix:/tmp/memcached.socket;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a server group.

**memcached_read_timeout**

**SYNTAX:** `memcached_read_timeout time;`

**DEFAULT:** 60s

**CONTEXT:** http, server, location

Defines a timeout for reading a response from the memcached server. A timeout is set only between two successive read operations, not for the transmission of the whole response. If a memcached server does not transmit anything within this time, the connection is closed.

**memcached_send_timeout**

**SYNTAX:** `memcached_send_timeout time;`

**DEFAULT:** 60s

**CONTEXT:** http, server, location

Sets a timeout for transmitting a request to the memcached server. A timeout is set only between two successive write operations, not for the transmission of the whole request. If a memcached server does not receive anything within this time, a connection is closed.
2.28 Module ngx_http_mp4_module

2.28.1 Summary

The ngx_http_mp4_module module provides pseudo-streaming server-side support for H.264/AAC files. Such files typically have the .mp4, .m4v, or .m4a filename extensions.

The pseudo-streaming works in alliance with a compatible Flash players. A player sends an HTTP request to the server with a start time specified in the query string argument (named simply start and specified in seconds), and the server responds with the stream such that its start position corresponds to the requested time, for example:

```
http://example.com/elephants_dream.mp4?start=238.88
```

This allows performing a random seeking at any time, or starting playback in the middle of the timeline.

To support seeking, H.264-based formats store the metadata in the so-called "moov atom." It is a part of the file that holds the index information for the whole file.

To start playback, a player first needs to read metadata. This is done by sending a special request with the start=0 argument. Much of encoding software will insert the metadata at the end of the file. This is bad for pseudo-streaming: the metadata should be located at the beginning of the file, or else the entire file will have to be downloaded to start playing. If a file is well-formed (with metadata at the beginning of a file), nginx just sends back the file contents. Otherwise, it has to read the file and prepare a new stream so that the metadata comes before the media data. This involves some CPU, memory, and disk I/O overhead, so it is a good idea to prepare an original file for pseudo-streaming, rather than having nginx do this on every such request.

For a matching request with a non-zero start argument, nginx will read the metadata from the file, prepare the stream starting from the requested offset, and send it to a client. This has the same overhead as described above.

If a matching request does not include the start argument, there is no overhead, and the file is just sent as a static resource. Some players also support byte-range requests, and thus do not require this module at all.

This module is not built by default, it should be enabled with the `--with-http_mp4_module` configuration parameter.

If a third-party mp4 module was previously used, it should be disabled.

A similar pseudo-streaming support for FLV files is provided by the ngx_http_flv_module module.

2.28.2 Example Configuration
location /video/ {
    mp4:
    mp4_buffer_size 1m;
    mp4_max_buffer_size 5m;
    mp4_limit_rate on;
    mp4_limit_rate_after 30s;
}

2.28.3 Directives

mp4

SYNTAX:  mp4;
DEFAULT —
CONTEXT: location

Turns on module processing in a surrounding location.

mp4_buffer_size

SYNTAX:  mp4_buffer_size size;
DEFAULT 512K
CONTEXT: http, server, location

Sets the initial size of a memory buffer used for processing MP4 files.

mp4_max_buffer_size

SYNTAX:  mp4_max_buffer_size size;
DEFAULT 10M
CONTEXT: http, server, location

During metadata processing, a larger buffer may become necessary. Its size cannot exceed the specified size, or else nginx will return the 500 Internal Server Error server error, and log the following message:

"/some/movie/file.mp4" mp4 moov atom is too large: 12583268, you may want to increase mp4_max_buffer_size

mp4_limit_rate

SYNTAX:  mp4_limit_rate on | off | factor;
DEFAULT off
CONTEXT: http, server, location

Enables or disables rate limiting based on the average bitrate of the MP4 file served. To calculate the rate, the bitrate is multiplied by the specified factor. The special value “on” corresponds to the factor of 1.1.
This directive is available as part of our *commercial subscription*.

**mp4_limit_rate_after**

**SYNTAX:** `mp4_limit_rate_after time;`

**DEFAULT:** `1m`

**CONTEXT:** `http, server, location`

Limits the rate after sending the specified amount of media data.

This directive is available as part of our *commercial subscription*.
2.29 Module ngx_http_perl_module

2.29.1 Summary

The ngx_http_perl_module module is used to implement location and variable handlers in Perl and insert Perl calls into SSI.

This module is not built by default, it should be enabled with the --with-http_perl_module configuration parameter.

This module requires Perl version 5.6.1 or higher. The C compiler should be compatible with the one used to build Perl.

2.29.2 Known Bugs

The module is experimental, caveat emptor applies.

In order for Perl to recompile the modified modules during reconfiguration, it should be built with the -Dusethreads=yes or -Dusethreads=yes parameters. Also, to make Perl leak less memory at run time, it should be built with the -Dusemymalloc=no parameter. To check the values of these parameters in an already built Perl (preferred values are specified in the example), run:

```
$ perl -V:usemultiplicity -V:usemymalloc
usemultiplicity = 'define';
usemymalloc = 'n';
```

Note that after rebuilding Perl with the new -Dusemultiplicity=yes or -Dusethreads=yes parameters, all binary Perl modules will have to be rebuilt as well — they will just stop working with the new Perl.

There is a possibility that the main process and then worker processes will grow in size after every reconfiguration. If the main process grows to an unacceptable size, the live upgrade procedure can be applied without changing the executable file.

While the Perl module is performing a long-running operation, such as resolving a domain name, connecting to another server, or querying a database, other requests assigned to the current worker process will not be processed. It is thus recommended to perform only such operations that have predictable and short execution time, such as accessing the local file system.

The issues mentioned below affect only the nginx versions before 0.6.22.

The $r request object methods return data only as a string value, and the value itself is stored in memory allocated by nginx from its own pools, not by Perl. This helps to reduce the number of copy operations involved in most cases; however it can lead to errors in some cases. For example, a worker process trying to use such data in the numeric context will terminate with an error (FreeBSD):
or (Linux):

```plaintext
*** glibc detected *** realloc(): invalid pointer: ... ***
Out of memory!
Callback called exit.
```

The workaround is simple — the method’s value should be assigned to a variable. For example, the following code

```perl
my $i = $r->variable('counter') + 1;
```

should be replaced by

```perl
my $i = $r->variable('counter');
$i++;
```

Since most strings inside nginx are stored without a terminating null character, they are similarly returned by the `$r` request object methods (except for the `$r->filename` and `$r->request_body_file` methods). Thus, such values cannot be used as filenames and the likes. The workaround is similar to the previous case — the value should either be assigned to a variable (this results in data copying and adding of the necessary null character) or used in an expression, for example:

```perl
open FILE, '/path/' . $r->variable('name');
```

### 2.29.3 Example Configuration

```perl
http {
    perl_modules perl/lib;
    perl_require hello.pm;
    perl_set $msie6
    sub {
        my $r = shift;
        my $ua = $r->header_in("User-Agent");
        return "" if $ua =~ /Opera/;
        return "1" if $ua =~ / MSIE [6-9]\./d+/;
        return "";
    }
}
```

Nginx, Inc. p.133 of 242
The `perl/lib/hello.pm` module:

```perl
package hello;
use nginx;
sub handler {
    my $r = shift;
    $r->send_http_header("text/html");
    return OK if $r->header_only;
    $r->print("hello!\n<br/>");
    if (-f $r->filename or -d _) {
        $r->print($r->uri, " exists!\n");
    }
    return OK;
}
1;
__END__
```

### 2.29.4 Directives

#### perl

**SYNTAX:** `perl module:function|sub { ... };`

**DEFAULT** —

**CONTEXT:** location, limit_except

Sets a Perl handler for the given location.

#### perl_modules

**SYNTAX:** `perl_modules path;`

**DEFAULT** —

**CONTEXT:** http

Sets an additional path for Perl modules.

#### perl_require

**SYNTAX:** `perl_require module;`

**DEFAULT** —

**CONTEXT:** http

Defines the name of a module that will be loaded during each reconfiguration. Several `perl_require` directives can be present.
perl_set
SYNTAX: perl_set $variable module::function'sub \{ ...\}';
DEFAULT —
CONTEXT: http
Installs a Perl handler for the specified variable.

2.29.5 Calling Perl from SSI
An SSI command calling Perl has the following format:

```perl
<!--# perl sub="module::function" arg="parameter1" arg="parameter2" ... -->
```

2.29.6 The $r Request Object Methods

$r->args
returns request arguments.

$r->filename
returns a filename corresponding to the request URI.

$r->has_request_body(handler)
returns 0 if there is no body in a request. If there is a body, the specified
handler is set for the request and 1 is returned. After reading the request
body, nginx will call the specified handler. Note that the handler function
should be passed by reference. Example:

```perl
package hello;
use nginx;
sub handler {
    my $r = shift;
    if ($r->request_method ne "POST") {
        return DECLINED;
    }
    if ($r->has_request_body(&post)) {
        return OK;
    }
    return HTTP_BAD_REQUEST;
}
sub post {
    my $r = shift;
    $r->send_http_header;
    $r->print("request_body: \"", $r->request_body, "\"<br/>");
    $r->print("request_body_file: \"", $r->request_body_file, "\"<br/>");
    return OK;
}
```

Nginx, Inc. p.135 of 242
$r->allow_ranges
enables the use of byte ranges when sending responses.

$r->discard_request_body
instructs nginx to discard the request body.

$r->header_in(field)
returns the value of the specified client request header field.

$r->header_only
determines whether the whole response or only its header should be sent to the client.

$r->header_out(field, value)
sets a value for the specified response header field.

$r->internal_redirect(uri)
does an internal redirect to the specified uri. An actual redirect happens after the Perl handler execution is completed.

Redirections to named locations are currently not supported.

$r->log_error(errno, message)
writes the specified message into the error_log. If errno is non-zero, an error code and its description will be appended to the message.

$r->print(text, ...)
passes data to a client.

$r->request_body
returns the client request body if it has not been written to a temporary file. To ensure that the client request body is in memory, its size should be limited by client_max_body_size, and a sufficient buffer size should be set using client_body_buffer_size.

$r->request_body_file
returns the name of the file with the client request body. After the processing, the file should be removed. To always write a request body to a file, client_body_in_file_only should be enabled.

$r->request_method
returns the client request HTTP method.

$r->remote_addr
returns the client IP address.

$r->flush
immediately sends data to the client.

$r->sendfile(name[, offset[, length]])
sends the specified file content to the client. Optional parameters specify the initial offset and length of the data to be transmitted. The actual data transmission happens after the Perl handler has completed.

$r->send_http_header([type])
sends the response header to the client. The optional type parameter sets the value of the Content-Type response header field. If the value is an empty string, the Content-Type header field will not be sent.

$r->status(code)
sets a response code.

$r->sleep(milliseconds, handler)
sets the specified handler and stops request processing for the specified time. In the mean time, nginx continues to process other requests. After the specified time has elapsed, nginx will call the installed handler. Note that the handler function should be passed by reference. In order to pass data between handlers, $r->variable() should be used. Example:

```perl
package hello;
use nginx;

sub handler {
    my $r = shift;
    $r->discard_request_body;
    $r->variable("var", "OK");
    $r->sleep(1000, &next);
    return OK;
}

sub next {
    my $r = shift;
    $r->send_http_header;
    $r->print($r->variable("var"));
    return OK;
}

1;
__END__
```

$r->unescape(text)
decodes a text encoded in the "%XX" form.

$r->uri
returns a request URI.

$r->variable(name[, value])
returns or sets the value of the specified variable. Variables are local to each request.
2.30 Module ngx_http_proxy_module

2.30.1 Summary

The ngx_http_proxy_module module allows passing requests to another server.

2.30.2 Example Configuration

```plaintext
location / {
    proxy_pass http://localhost:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

2.30.3 Directives

**proxy_bind**

**SYNTAX:** `proxy_bind address | off;`

**DEFAULT** —

**CONTEXT:** http, server, location

This directive appeared in version 0.8.22.

Makes outgoing connections to a proxied server originate from the specified local IP address. Parameter value can contain variables (1.3.12). The special value `off` (1.3.12) cancels the effect of the `proxy_bind` directive inherited from the previous configuration level, which allows the system to auto-assign the local IP address.

**proxy_buffer_size**

**SYNTAX:** `proxy_buffer_size size;`

**DEFAULT** `4k|8k`

**CONTEXT:** http, server, location

Sets the size of the buffer used for reading the first part of a response received from the proxied server. This part usually contains a small response header. By default, the buffer size is equal to the size of one buffer set by the `proxy_buffers` directive. It can be made smaller however.

**proxy_buffering**

**SYNTAX:** `proxy_buffering on | off;`

**DEFAULT** `on`

**CONTEXT:** http, server, location

Enables or disables buffering of responses from the proxied server.
When buffering is enabled, nginx receives a response from the proxied server as soon as possible, saving it into the buffers set by the `proxy_buffer_size` and `proxy_buffers` directives. If the whole response does not fit into memory, a part of it can be saved to a temporary file on the disk. Writing to temporary files is controlled by the `proxy_max_temp_file_size` and `proxy_temp_file_write_size` directives.

When buffering is disabled, a response is passed to a client synchronously, immediately as it is received. nginx will not try to read the whole response from the proxied server. The maximum size of the data that nginx can receive from the server at a time is set by the `proxy_buffer_size` directive.

Buffering can also be enabled or disabled by passing “yes” or “no” in the `X-Accel-Buffering` response header field. This capability can be disabled using the `proxy_ignore_headers` directive.

**proxy_buffers**

**SYNTAX:**
```
proxy_buffers number size;
```

**DEFAULT:** `8 4k|8k`

**CONTEXT:** `http`, `server`, `location`

Sets the `number` and `size` of buffers used for reading a response from the proxied server, for a single connection. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.

**proxy_busy_buffers_size**

**SYNTAX:**
```
proxy_busy_buffers_size size;
```

**DEFAULT:** `8k|16k`

**CONTEXT:** `http`, `server`, `location`

When buffering of responses from the proxied server is enabled, limits the total `size` of buffers that can be busy sending a response to the client while the response is not yet fully read. In the mean time, the rest of the buffers can be used for reading a response and, if needed, buffering part of a response to a temporary file. By default, `size` is limited by the size of two buffers set by the `proxy_buffer_size` and `proxy_buffers` directives.

**proxy_cache**

**SYNTAX:**
```
proxy_cache zone | off;
```

**DEFAULT:** `off`

**CONTEXT:** `http`, `server`, `location`

Defines a shared memory zone used for caching. The same zone can be used in several places. The `off` parameter disables caching inherited from the previous configuration level.
proxy_cache_bypass

SYNTAX:   proxy_cache_bypass string...;
DEFAULT    —
CONTEXT:  http, server, location

Defines conditions under which the response will not be taken from a cache. If at least one value of the string parameters is not empty and is not equal to "0" then the response will not be taken from the cache:

```nginx
proxy_cache_bypass $cookie_nocache $arg_nocache $arg_comment;
proxy_cache_bypass $http_pragma $http_authorization;
```

Can be used along with the proxy_no_cache directive.

proxy_cache_key

SYNTAX:   proxy_cache_key string;
DEFAULT   $scheme$proxy_host$request_uri
CONTEXT:  http, server, location

Defines a key for caching, for example

```nginx
proxy_cache_key "$host$request_uri $cookie_user";
```

By default, the directive’s value is close to the string

```nginx
proxy_cache_key $scheme$proxy_host$uri$is_args$args;
```

proxy_cache_lock

SYNTAX:   proxy_cache_lock on | off;
DEFAULT   off
CONTEXT:  http, server, location
This directive appeared in version 1.1.12.

When enabled, only one request at a time will be allowed to populate a new cache element identified according to the proxy_cache_key directive by passing a request to a proxied server. Other requests of the same cache element will either wait for a response to appear in the cache or the cache lock for this element to be released, up to the time set by the proxy_cache_lock_timeout directive.

proxy_cache_lock_timeout

SYNTAX:   proxy_cache_lock_timeout time;
DEFAULT   5s
CONTEXT:  http, server, location
This directive appeared in version 1.1.12.

Sets a timeout for proxy_cache_lock.
**proxy_cache_methods**

**SYNTAX:**  
`proxy_cache_methods GET | HEAD | POST ...;`

**DEFAULT:**  
`GET HEAD`

**CONTEXT:**  
http, server, location

This directive appeared in version 0.7.59.

If the client request method is listed in this directive then the response will be cached. “GET” and “HEAD” methods are always added to the list, though it is recommended to specify them explicitly. See also the `proxy_no_cache` directive.

**proxy_cache_min_uses**

**SYNTAX:**  
`proxy_cache_min_uses number;`

**DEFAULT:**  
1

**CONTEXT:**  
http, server, location

Sets the `number` of requests after which the response will be cached.

**proxy_cache_path**

**SYNTAX:**  
`proxy_cache_path path [levels=levels] keys_zone=name:size`  
`[inactive=time] [max_size=size] [loader_files=number]`  
`[loader_sleep=time] [loader_threshold=time];`

**DEFAULT:**  
—

**CONTEXT:**  
http

Sets the path and other parameters of a cache. Cache data are stored in files. Both the key and file name in a cache are a result of applying the MD5 function to the proxied URL.

The `levels` parameter defines hierarchy levels of a cache. For example, in the following configuration

```plaintext
proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=one:10m;
```

file names in a cache will look like this:

```
/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

A cached response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the cache can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both cache and a directory holding temporary files, set by the `proxy_temp_path` directive, are put on the same file system.

In addition, all active keys and information about data are stored in a shared memory zone, whose `name` and `size` are configured by the `keys_zone` parameter. Cached data that are not accessed during the time specified by the
inactive parameter get removed from the cache regardless of their freshness. By default, inactive is set to 10 minutes.

The special “cache manager” process monitors the maximum cache size set by the max_size parameter. When this size is exceeded, it removes the least recently used data.

A minute after the start the special “cache loader” process is activated. It loads information about previously cached data stored on file system into a cache zone. The loading is done in iterations. During one iteration no more than loader_files items are loaded (by default, 100). Besides, the duration of one iteration is limited by the loader_threshold parameter (by default, 200 milliseconds). Between iterations, a pause configured by the loader_sleep parameter (by default, 50 milliseconds) is made.

proxy_cache_purge

SYNTAX:  proxy_cache_purge string ...;

DEFAULT —

CONTEXT:  http, server, location

This directive appeared in version 1.5.7.

Defines conditions under which the request will be considered a cache purge request. If at least one value of the string parameters is not empty and is not equal to “0” then the cache entry with a corresponding cache key is removed. The result of successful operation is indicated by returning the 204 No Content response.

If the cache key of a purge request ends with an asterisk (“*”), all cache entries matching the wildcard key will be removed from the cache.

Example configuration:

```bash
proxy_cache_path /data/nginx/cache keys_zone=cache_zone:10m;
map $request_method $purge_method {
    PURGE 1;
    default 0;
}
server {
    ...
    location / {
        proxy_pass http://backend;
        proxy_cache cache_zone;
        proxy_cache_key $uri;
        proxy_cache_purge $purge_method;
    }
}
```

This functionality is available as part of our commercial subscription.
proxy_cache_revalidate
SYNTAX: \texttt{proxy_cache_revalidate on | off};
DEFAULT \texttt{off}
CONTEXT: \texttt{http, server, location}
This directive appeared in version 1.5.7.

Enables revalidation of expired cache items using conditional requests with the \textit{If-Modified-Since} header field.

\textbf{proxy_cache_use_stale}
SYNTAX: \texttt{proxy_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | off ...};
DEFAULT \texttt{off}
CONTEXT: \texttt{http, server, location}

Determines in which cases a stale cached response can be used when an error occurs during communication with the proxied server. The directive’s parameters match the parameters of the \texttt{proxy_next_upstream} directive.

Additionally, the \texttt{updating} parameter permits using a stale cached response if it is currently being updated. This allows minimizing the number of accesses to proxied servers when updating cached data.

To minimize the number of accesses to proxied servers when populating a new cache element, the \texttt{proxy_cache_lock} directive can be used.

\textbf{proxy_cache_valid}
SYNTAX: \texttt{proxy_cache_valid \texttt{[code ...]} time};
DEFAULT \texttt{—}
CONTEXT: \texttt{http, server, location}

Sets caching time for different response codes. For example, the following directives

\begin{verbatim}
proxy_cache_valid 200 302 10m;
proxy_cache_valid 404 1m;
\end{verbatim}

set 10 minutes of caching for responses with codes 200 and 302 and 1 minute for responses with code 404.

If only caching \texttt{time} is specified

\begin{verbatim}
proxy_cache_valid 5m;
\end{verbatim}

then only 200, 301, and 302 responses are cached.

In addition, the \texttt{any} parameter can be specified to cache any responses:

\begin{verbatim}
proxy_cache_valid 200 302 10m;
proxy_cache_valid 301 1h;
proxy_cache_valid any 1m;
\end{verbatim}
Parameters of caching can also be set directly in the response header. This has higher priority than setting of caching time using the directive. The X-Accel-Expires header field sets caching time of a response in seconds. The zero value disables caching for a response. If a value starts with the @ prefix, it sets an absolute time in seconds since Epoch, up to which the response may be cached. If header does not include the X-Accel-Expires field, parameters of caching may be set in the header fields Expires or Cache-Control. If a header includes the Set-Cookie field, such a response will not be cached. Processing of one or more of these response header fields can be disabled using the proxy-ignore_headers directive.

**proxy_connect_timeout**

**SYNTAX:**

```
proxy_connect_timeout time;
```

**DEFAULT:** 60s

**CONTEXT:** http, server, location

Defines a timeout for establishing a connection with a proxied server. It should be noted that this timeout cannot usually exceed 75 seconds.

**proxy_cookie_domain**

**SYNTAX:**

```
proxy_cookie_domain off;
```

**SYNTAX:**

```
proxy_cookie_domain domain replacement;
```

**DEFAULT:** off

**CONTEXT:** http, server, location

This directive appeared in version 1.1.15.

Sets a text that should be changed in the domain attribute of the Set-Cookie header fields of a proxied server response. Suppose a proxied server returned the Set-Cookie header field with the attribute “domain=localhost”. The directive

```
proxy_cookie_domain localhost example.org;
```

will rewrite this attribute to “domain=example.org”. A dot at the beginning of the domain and replacement strings and the domain attribute is ignored. Matching is case-insensitive.

The domain and replacement strings can contain variables:

```
proxy_cookie_domain www.$host $host;
```

The directive can also be specified using regular expressions. In this case, domain should start from the “~” symbol. A regular expression can contain named and positional captures, and replacement can reference them:

```
proxy_cookie_domain ~(?P<sl_domain>[-0-9a-z]+\.[a-z]+)$ $sl_domain;
```

There could be several `proxy_cookie_domain` directives.
The off parameter cancels the effect of all proxy_cookie_domain directives on the current level:

```
proxy_cookie_domain off;
proxy_cookie_domain localhost example.org;
proxy_cookie_domain www.example.org example.org;
```

**proxy_cookie_path**

**SYNTAX:** proxy_cookie_path off;
**DEFAULT** off
**CONTEXT:** http, server, location

This directive appeared in version 1.1.15.

Sets a text that should be changed in the path attribute of the Set-Cookie header fields of a proxied server response. Suppose a proxied server returned the Set-Cookie header field with the attribute “path=/two/some/uri/”. The directive

```
proxy_cookie_path /two/ /;
```

will rewrite this attribute to “path=/some/uri/”.

The path and replacement strings can contain variables:

```
proxy_cookie_path $uri /some$uri;
```

The directive can also be specified using regular expressions. In this case, path should either start from the “~” symbol for a case-sensitive matching, or from the “~*” symbols for case-insensitive matching. A regular expression can contain named and positional captures, and replacement can reference them:

```
proxy_cookie_path ~*/user/([^/]+) /u/$1;
```

There could be several proxy_cookie_path directives:

```
proxy_cookie_path /one/ /;
proxy_cookie_path /two/ /;
```

The off parameter cancels the effect of all proxy_cookie_path directives on the current level:

```
proxy_cookie_path off;
proxy_cookie_path /two/ /;
proxy_cookie_path ~*/user/(.*) /u/$1;
```
proxy_headers_hash_bucket_size
SYNTAX:  proxy_headers_hash_bucket_size size;
DEFAULT 64
CONTEXT: http, server, location

Sets the bucket size for hash tables used by the proxy_hide_header and proxy_set_header directives. The details of setting up hash tables are provided in a separate document.

proxy_headers_hash_max_size
SYNTAX:  proxy_headers_hash_max_size size;
DEFAULT 512
CONTEXT: http, server, location

Sets the maximum size of hash tables used by the proxy_hide_header and proxy_set_header directives. The details of setting up hash tables are provided in a separate document.

proxy_hide_header
SYNTAX:  proxy_hide_header field;
DEFAULT —
CONTEXT: http, server, location

By default, nginx does not pass the header fields Date, Server, X-Pad, and X-Accel-... from the response of a proxied server to a client. The proxy_hide_header directive sets additional fields that will not be passed. If, on the contrary, the passing of fields needs to be permitted, the proxy_pass_header directive can be used.

proxy_http_version
SYNTAX:  proxy_http_version 1.0 | 1.1;
DEFAULT 1.0
CONTEXT: http, server, location
This directive appeared in version 1.1.4.

Sets the HTTP protocol version for proxying. By default, version 1.0 is used. Version 1.1 is recommended for use with keepalive connections.

proxy_ignore_client_abort
SYNTAX:  proxy_ignore_client_abort on | off;
DEFAULT off
CONTEXT: http, server, location

Determines whether the connection with a proxied server should be closed when a client closes a connection without waiting for a response.
proxy_ignore_headers
SYNTAX: proxy_ignore_headers field...;
DEFAULT —
CONTEXT: http, server, location

Disables processing of certain response header fields from the proxied server. The following fields can be ignored: X-Accel-Redirect, X-Accel-Expires, X-Accel-Limit-Rate (1.1.6), X-Accel-Buffering (1.1.6), X-Accel-Charset (1.1.6), Expires, Cache-Control, and Set-Cookie (0.8.44).

If not disabled, processing of these header fields has the following effect:

• X-Accel-Expires, Expires, Cache-Control, and Set-Cookie set the parameters of response caching;
• X-Accel-Redirect performs an internal redirect to the specified URI;
• X-Accel-Limit-Rate sets the rate limit for transmission of a response to a client;
• X-Accel-Buffering enables or disables buffering of a response;
• X-Accel-Charset sets the desired charset of a response.

proxy_intercept_errors
SYNTAX: proxy_intercept_errors on | off;
DEFAULT off
CONTEXT: http, server, location

Determines whether proxied responses with codes greater than or equal to 300 should be passed to a client or be redirected to nginx for processing with the error_page directive.

proxy_max_temp_file_size
SYNTAX: proxy_max_temp_file_size size;
DEFAULT 1024m
CONTEXT: http, server, location

When buffering of responses from the proxied server is enabled, and the whole response does not fit into the memory buffers set by the proxy_buffer_size and proxy_buffers directives, a part of the response can be saved to a temporary file. This directive sets the maximum size of a temporary file. The size of data written to a temporary file at a time is set by the proxy_temp_file_write_size directive.

The zero value disables buffering of responses to temporary files.
**proxy_method**

**SYNTAX:**  
proxy_method method;

**DEFAULT** —

**CONTEXT:** http, server, location

Specifies the HTTP *method* to use in requests forwarded to the proxied server instead of the method from the client request.

**proxy_next_upstream**

**SYNTAX:**  
proxy_next_upstream error | timeout | invalid_header | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | off ...;

**DEFAULT** error timeout

**CONTEXT:** http, server, location

Specifies in which cases a request should be passed to the next server:

- **error**  
  an error occurred while establishing a connection with the server, passing a request to it, or reading the response header;

- **timeout**  
  a timeout has occurred while establishing a connection with the server, passing a request to it, or reading the response header;

- **invalid_header**  
  a server returned an empty or invalid response;

- **http_500**  
  a server returned a response with the code 500;

- **http_502**  
  a server returned a response with the code 502;

- **http_503**  
  a server returned a response with the code 503;

- **http_504**  
  a server returned a response with the code 504;

- **http_403**  
  a server returned a response with the code 403;

- **http_404**  
  a server returned a response with the code 404;

- **off**  
  disables passing a request to the next server.

One should bear in mind that passing a request to the next server is only possible if nothing has been sent to a client yet. That is, if an error or timeout occurs in the middle of the transferring of a response, fixing this is impossible.

The directive also defines what is considered an unsuccessful attempt of communication with a *server*. The cases of **error**, **timeout** and **invalid_header** are always considered unsuccessful attempts, even if they are not specified in the directive. The cases of **http_500**, **http_502**, **http_503** and
http_504 are considered unsuccessful attempts only if they are specified in the directive. The cases of http_403 and http_404 are never considered unsuccessful attempts.

**proxy_no_cache**

**SYNTAX:**  
proxy_no_cache string ...;

**DEFAULT** —

**CONTEXT:** http, server, location

Defines conditions under which the response will not be saved to a cache. If at least one value of the string parameters is not empty and is not equal to “0” then the response will not be saved:

```nginx
proxy_no_cache $cookie_nocache $arg_nocache$arg_comment ;
proxy_no_cache $http_pragma $http_authorization ;
```

Can be used along with the proxy_cache_bypass directive.

**proxy_pass**

**SYNTAX:**  
proxy_pass URL;

**DEFAULT** —

**CONTEXT:** location, if in location, limit_except

Sets the protocol and address of a proxied server and an optional URI to which a location should be mapped. As a protocol, “http” or “https” can be specified. The address can be specified as a domain name or IP address, and an optional port:

```nginx
proxy_pass http://localhost:8000/ uri /;
```

or as a UNIX-domain socket path specified after the word “unix” and enclosed in colons:

```nginx
proxy_pass http://unix:/tmp/backend.socket:/uri/;
```

If a domain name resolves to several addresses, all of them will be used in a round-robin fashion. In addition, an address can be specified as a server group.

A request URI is passed to the server as follows:

- If the proxy_pass directive is specified with a URI, then when a request is passed to the server, the part of a normalized request URI matching the location is replaced by a URI specified in the directive:

```nginx
location /name/ {
    proxy_pass http://127.0.0.1/remote/;
}
```
• If `proxy_pass` is specified without a URI, the request URI is passed to
the server in the same form as sent by a client when the original request is
processed, or the full normalized request URI is passed when processing
the changed URI:

```nginx
location /some/path/ {
    proxy_pass http://127.0.0.1;
}
```

Before version 1.1.12, if `proxy_pass` is specified without a URI, the
original request URI might be passed instead of the changed URI in
some cases.

In some cases, the part of a request URI to be replaced cannot be
determined:

• When location is specified using a regular expression.
  In this case, the directive should be specified without a URI.

• When the URI is changed inside a proxied location using the `rewrite`
directive, and this same configuration will be used to process a request
  (`break`):

```nginx
location /name/ {
    rewrite /name/([^/]*) /users?name=$1 break;
    proxy_pass http://127.0.0.1;
}
```

In this case, the URI specified in the directive is ignored and the full
changed request URI is passed to the server.

A server name, its port and the passed URI can also be specified using
variables:

```nginx
proxy_pass http://$host$uri;
```

or even like this:

```nginx
proxy_pass $request;
```

In this case, the server name is searched among the described server
groups, and, if not found, is determined using a resolver.

WebSocket proxying requires special configuration and is supported since
version 1.3.13.
proxy_pass_header

SYNTAX:  proxy_pass_header field;
DEFAULT  —
CONTEXT: http, server, location

Permits passing otherwise disabled header fields from a proxied server to a client.

proxy_read_timeout

SYNTAX:  proxy_read_timeout time;
DEFAULT  60s
CONTEXT: http, server, location

Defines a timeout for reading a response from the proxied server. A timeout is set only between two successive read operations, not for the transmission of the whole response. If a proxied server does not transmit anything within this time, a connection is closed.

proxy_pass_request_body

SYNTAX:  proxy_pass_request_body on | off;
DEFAULT  on
CONTEXT: http, server, location

Indicates whether the original request body is passed to the proxied server.

```location /x-accel-redirect-here/ {
  proxy_method GET;
  proxy_pass_request_body off;
  proxy_set_header Content-Length "";
  proxy_pass ...
}
```

See also the proxy_set_header and proxy_pass_request_headers directives.

proxy_pass_request_headers

SYNTAX:  proxy_pass_request_headers on | off;
DEFAULT  on
CONTEXT: http, server, location

Indicates whether the header fields of the original request are passed to the proxied server.

```location /x-accel-redirect-here/ {
  proxy_method GET;
  proxy_pass_request_headers off;
  proxy_pass_request_body off;
  proxy_pass ...
}
```
See also the `proxy_set_header` and `proxy_pass_request_body` directives.

**proxy_redirect**

**SYNTAX:**
```
proxy_redirect default;
```

**SYNTAX:**
```
proxy_redirect off;
```

**SYNTAX:**
```
proxy_redirect redirect replacement;
```

**DEFAULT** `default`

**CONTEXT:** `http`, server, location

Sets the text that should be changed in the `Location` and `Refresh` header fields of a proxied server response. Suppose a proxied server returned the header field “Location: http://localhost:8000/two/some/uri/”. The directive

```
```

will rewrite this string to “Location: http://frontend/one/some/uri/”.

A server name may be omitted in the `replacement` string:

```
proxy_redirect http://localhost:8000/two/ /;
```

then the primary server’s name and port, if different from 80, will be inserted.

The default replacement specified by the `default` parameter uses the parameters of the `location` and `proxy_pass` directives. Hence, the two configurations below are equivalent:

```
location /one/ {
    proxy_pass http://upstream:port/two/;
    proxy_redirect default;
}
```

```
location /one/ {
    proxy_pass http://upstream:port/two/;
    proxy_redirect http://upstream:port/two/ /one/;
}
```

The `default` parameter is not permitted if `proxy_pass` is specified using variables.

A `replacement` string can contain variables:

```
```

A `redirect` can also contain (1.1.11) variables:

```
proxy_redirect http://$proxy_host:8000/ /;
```

The directive can be specified (1.1.11) using regular expressions. In this case, `redirect` should either start with the “~” symbol for a case-sensitive matching, or with the “~*” symbols for case-insensitive matching. A regular
expression can contain named and positional captures, and replacement can reference them:

```bash
proxy_redirect ~^(http://[\^:]+)\d+(/.+) $1$2;
proxy_redirect ~*#/user/([^/]*)/(.*) http://$1.example.com/$2;
```

There could be several `proxy_redirect` directives:

```bash
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

The `off` parameter cancels the effect of all `proxy_redirect` directives on the current level:

```bash
proxy_redirect off;
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

Using this directive, it is also possible to add host names to relative redirects issued by a proxied server:

```bash
proxy_redirect / /;
```

**proxy_send_lowat**

**SYNTAX:** `proxy_send_lowat size;`

**DEFAULT** 0

**CONTEXT:** http, server, location

If the directive is set to a non-zero value, nginx will try to minimize the number of send operations on outgoing connections to a proxied server by using either `NOTE_LOWAT` flag of the kqueue method, or the `SO_SNDLOWAT` socket option, with the specified size.

This directive is ignored on Linux, Solaris, and Windows.

**proxy_send_timeout**

**SYNTAX:** `proxy_send_timeout time;`

**DEFAULT** 60s

**CONTEXT:** http, server, location

Sets a timeout for transmitting a request to the proxied server. A timeout is set only between two successive write operations, not for the transmission of the whole request. If a proxied server does not receive anything within this time, a connection is closed.
CHAPTER 2. HTTP SERVER MODULES

2.30. MODULE NGX_HTTP_PROXY_MODULE

proxy_set_body

SYNTAX:    proxy_set_body value;
DEFAULT     —
CONTEXT:   http, server, location

Allows redefining the request body passed to the proxied server. A value can contain text, variables, and their combination.

proxy_set_header

SYNTAX:    proxy_set_header field value;
DEFAULT Host $proxy_host
DEFAULT Connection close
CONTEXT:   http, server, location

Allows redefining or appending fields to the request header passed to the proxied server. A value can contain text, variables, and their combinations. These directives are inherited from the previous level if and only if there are no proxy_set_header directives defined on the current level. By default, only two fields are redefined:

```
proxy_set_header Host $proxy_host;
proxy_set_header Connection close;
```

An unchanged Host request header field can be passed like this:

```
proxy_set_header Host $http_host;
```

However, if this field is not present in a client request header then nothing will be passed. In such a case it is better to use the $host variable - its value equals the server name in the Host request header field or the primary server name if this field is not present:

```
proxy_set_header Host $host;
```

In addition, the server name can be passed together with the port of the proxied server:

```
proxy_set_header Host $host:$proxy_port;
```

If the value of a header field is an empty string then this field will not be passed to a proxied server:

```
proxy_set_header Accept-Encoding "";
```

proxy_ssl_ciphers

SYNTAX:    proxy_ssl_ciphers ciphers;
DEFAULT DEFINED
CONTEXT:   http, server, location
This directive appeared in version 1.5.6.

Specifies the enabled ciphers for requests to a proxied HTTPS server. The ciphers are specified in the format understood by the OpenSSL library. The full list can be viewed using the “openssl ciphers” command.

**proxy_ssl_session_reuse**

**SYNTAX:**  
`proxy_ssl_session_reuse on | off;`

**DEFAULT**  
on

**CONTEXT:** http, server, location

Determines whether SSL sessions can be reused when working with the proxied server. If the errors “SSL3_GET_FINISHED:digest check failed” appear in the logs, try disabling session reuse.

**proxy_ssl_protocols**

**SYNTAX:**  
`proxy_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];`

**DEFAULT**  
SSLv3 TLSv1 TLSv1.1 TLSv1.2

**CONTEXT:** http, server, location

This directive appeared in version 1.5.6.

Enables the specified protocols for requests to a proxied HTTPS server.

**proxy_store**

**SYNTAX:**  
`proxy_store on | off | string;`

**DEFAULT**  
on

**CONTEXT:** http, server, location

Enables saving of files to a disk. The on parameter saves files with paths corresponding to the directives alias or root. The off parameter disables saving of files. In addition, the file name can be set explicitly using the string with variables:

```
proxy_store /data/www$original_uri;
```

The modification time of files is set according to the received Last-Modified response header field. A response is first written to a temporary file, and then the file is renamed. Starting from version 0.8.9, temporary files and the persistent store can be put on different file systems. However, be aware that in this case a file is copied across two file systems instead of the cheap renaming operation. It is thus recommended that for any given location both saved files and a directory holding temporary files, set by the proxy_temp_path directive, are put on the same file system.

This directive can be used to create local copies of static unchangeable files, e.g.:

```
location /images/ {
```

Nginx, Inc.   p.155 of 242
CHAPTER 2. HTTP SERVER MODULES

2.30. MODULE NGX_HTTP_PROXY_MODULE

```plaintext
root /data/www;
error_page 404 = /fetch$uri;
}

location /fetch/ {
  internal;
  proxy_pass http://backend/;
  proxy_store on;
  proxy_store_access user:rw group:rw all:r;
  proxy_temp_path /data/temp;
  alias /data/www/;
}

or like this:

location /images/ {
  root /data/www;
  error_page 404 = @fetch;
}

location @fetch {
  internal;
  proxy_pass http://backend/;
  proxy_store on;
  proxy_store_access user:rw group:rw all:r;
  proxy_temp_path /data/temp;
  root /data/www/;
}
```

**proxy_store_access**

**SYNTAX:**

```
proxy_store_access users:permissions ...;
```

**DEFAULT**

```
user:rw
```

**CONTEXT:**

http, server, location

Sets access permissions for newly created files and directories, e.g.:

```
proxy_store_access user:rw group:rw all:r;
```

If any group or all access permissions are specified then user permissions may be omitted:

```
proxy_store_access group:rw all:r;
```

**proxy_temp_file_write_size**

**SYNTAX:**

```
proxy_temp_file_write_size size;
```

**DEFAULT**

```
8k|16k
```

**CONTEXT:**

http, server, location

Limits the size of data written to a temporary file at a time, when buffering of responses from the proxied server to temporary files is enabled. By default,
size is limited by two buffers set by the `proxy_buffer_size` and `proxy_buffers` directives. The maximum size of a temporary file is set by the `proxy_max_temp_file_size` directive.

**proxy_temp_path**

**Syntax:** `proxy_temp_path path [level1 [level2 [level3]]];`

**Default:** `proxy_temp`

**Context:** `http`, `server`, `location`

Defines a directory for storing temporary files with data received from proxied servers. Up to three-level subdirectory hierarchy can be used underneath the specified directory. For example, in the following configuration

```bash
proxy_temp_path /spool/nginx/proxy_temp 1 2;
```

a temporary file might look like this:

```bash
/spool/nginx/proxy_temp/7/45/00000123457
```

### 2.30.4 Embedded Variables

The `ngx_http_proxy_module` module supports embedded variables that can be used to compose headers using the `proxy_set_header` directive:

- `$proxy_host`
  - name and port of a proxied server;
- `$proxy_port`
  - port of a proxied server;
- `$proxy_add_x_forwarded_for`
  - the `X-Forwarded-For` client request header field with the `$remote_addr` variable appended to it, separated by a comma. If the `X-Forwarded-For` field is not present in the client request header, the `$proxy_add_x_forwarded_for` variable is equal to the `$remote_addr` variable.
2.31 Module ngx_http_random_index_module

2.31.1 Summary

The `ngx_http_random_index_module` module processes requests ending with the slash character (`/`) and picks a random file in a directory to serve as an index file. The module is processed before the `ngx_http_index_module` module.

This module is not built by default, it should be enabled with the `--with-http_random_index_module` configuration parameter.

2.31.2 Example Configuration

```plaintext
location / {
    random_index on;
}
```

2.31.3 Directives

`random_index`

**SYNTAX:** `random_index on | off;`
**DEFAULT:** `off`
**CONTEXT:** `location`

Enables or disables module processing in a surrounding location.
2.32 Module ngx_http_realip_module

2.32.1 Summary

The ngx_http_realip_module module is used to change the client address to the one sent in the specified header field.

This module is not built by default, it should be enabled with the --with-http_realip_module configuration parameter.

2.32.2 Example Configuration

```
set_real_ip_from 192.168.1.0/24;
set_real_ip_from 192.168.2.1;
set_real_ip_from 2001:0db8::/32;
real_ip_header X-Forwarded-For;
real_ip_recursive on;
```

2.32.3 Directives

**set_real_ip_from**

SYNTAX: `set_real_ip_from address | CIDR | unix:;

DEFAULT —

CONTEXT: http, server, location

Defines trusted addresses that are known to send correct replacement addresses. If the special value `unix:` is specified, all UNIX-domain sockets will be trusted.

IPv6 addresses are supported starting from versions 1.3.0 and 1.2.1.

**real_ip_header**

SYNTAX: `real_ip_header field | X-Real-IP | X-Forwarded-For | proxy_protocol;

DEFAULT X-Real-IP

CONTEXT: http, server, location

Defines a request header field used to send the address for a replacement.

The `proxy_protocol` parameter (1.5.12) changes the client address to the one from the PROXY protocol header. The PROXY protocol must be previously enabled by setting the `proxy_protocol` parameter in the `listen` directive.
real_ip_recursive

SYNTAX:  real_ip_recursive on | off;
DEFAULT off
CONTEXT: http, server, location
This directive appeared in versions 1.3.0 and 1.2.1.

If recursive search is disabled, the original client address that matches one of the trusted addresses is replaced by the last address sent in the request header field defined by the real_ip_header directive. If recursive search is enabled, the original client address that matches one of the trusted addresses is replaced by the last non-trusted address sent in the request header field.
2.33 Module ngx_http_referer_module

2.33.1 Summary

The ngx_http_referer_module module is used to block access to a site for requests with invalid values in the Referer header field. It should be kept in mind that fabricating a request with an appropriate Referer field value is quite easy, and so the intended purpose of this module is not to block such requests thoroughly but to block the mass flow of requests sent by regular browsers. It should also be taken into consideration that regular browsers may not send the Referer field even for valid requests.

2.33.2 Example Configuration

```nginx
valid_referers none blocked server_names
*.example.com example.* www.example.org/galleries/`
\google\;
if ($invalid_referer) {
  return 403;
}
```

2.33.3 Directives

**referer_hash_bucket_size**

**SYNTAX:** referer_hash_bucket_size size;

**DEFAULT:** 64

**CONTEXT:** server, location

This directive appeared in version 1.0.5.

Sets the bucket size for the valid referers hash tables. The details of setting up hash tables are provided in a separate document.

**referer_hash_max_size**

**SYNTAX:** referer_hash_max_size size;

**DEFAULT:** 2048

**CONTEXT:** server, location

This directive appeared in version 1.0.5.

Sets the maximum size of the valid referers hash tables. The details of setting up hash tables are provided in a separate document.

**valid_referers**

**SYNTAX:** valid_referers none | blocked | server_names | string ...;

**DEFAULT:**

**CONTEXT:** server, location
Specifies the Referer request header field values that will cause the embedded $invalid_referer variable to be set to an empty string. Otherwise, the variable will be set to “1”. Search for a match is case-insensitive.

Parameters can be as follows:

- **none**
  - the Referer field is missing in the request header;

- **blocked**
  - the Referer field is present in the request header, but its value has been deleted by a firewall or proxy server; such values are strings that do not start with “http://” or “https://”;

- **server_names**
  - the Referer request header field contains one of the server names;

- **arbitrary string**
  - defines a server name and an optional URI prefix. A server name can have an “*” at the beginning or end. During the checking, the server’s port in the Referer field is ignored;

- **regular expression**
  - the first symbol should be a “~”. It should be noted that an expression will be matched against the text starting after the “http://” or “https://”.

Example:

```conf
valid_referers none blocked server_names
  *.example.com example.* www.example.org/galleries/
  \.google\;
```
2.34 Module ngx_http_rewrite_module

2.34.1 Summary

The ngx_http_rewrite_module module is used to change request URI using regular expressions, return redirects, and conditionally select configurations.

The ngx_http_rewrite_module module directives are processed in the following order:

- the directives of this module specified on the server level are executed sequentially;
- repeatedly:
  - a location is searched based on a request URI;
  - the directives of this module specified inside the found location are executed sequentially;
  - the loop is repeated if a request URI was rewritten, but not more than 10 times.

2.34.2 Directives

break

SYNTAX: \texttt{break;}

DEFAULT —

CONTEXT: server, location, if

Stops processing the current set of ngx_http_rewrite_module directives. If a directive is specified inside the location, further processing of the request continues in this location.

Example:

\begin{verbatim}
if ($slow) {
    limit_rate 10k;
    break;
}
\end{verbatim}

if

SYNTAX: if (condition) \{ ... \}

DEFAULT —

CONTEXT: server, location

The specified condition is evaluated. If true, this module directives specified inside the braces are executed, and the request is assigned the configuration inside the if directive. Configurations inside the if directives are inherited from the previous configuration level.

A condition may be any of the following:
- a variable name; false if the value of a variable is an empty string or "0";

Before version 1.0.1, any string starting with "0" was considered a false value.

- comparison of a variable with a string using the "=" and "!=" operators;

- matching of a variable against a regular expression using the "=~" (for case-sensitive matching) and "=~*" (for case-insensitive matching) operators. Regular expressions can contain captures that are made available for later reuse in the $1..$9 variables. Negative operators "!~" and "!~*" are also available. If a regular expression includes the "}" or ";" characters, the whole expressions should be enclosed in single or double quotes.

- checking of a file existence with the "-f" and "!-f" operators;

- checking of a directory existence with the "-d" and "!-d" operators;

- checking of a file, directory, or symbolic link existence with the "-e" and "!-e" operators;

- checking for an executable file with the "-x" and "!-x" operators.

Examples:

```
if ( $http_user_agent ~ MSIE ) {
    rewrite "(.*)" /msie/$1 break;
}

if ( $http_cookie ~* "id=(\[^;]+) (?:;| $)"") {
    set $id $1;
}

if ( $request_method = POST) {
    return 405;
}

if ($slow) {
    limit_rate 10k;
}

if ($invalid_referer) {
    return 403;
}
```

A value of the $invalid_referer embedded variable is set by the valid_referers directive.

**return**

**SYNTAX:** return code [text];

**SYNTAX:** return code URL;

**SYNTAX:** return URL;

**DEFAULT** —

**CONTEXT:** server, location, if
Stops processing and returns the specified code to a client. The non-standard code 444 closes a connection without sending a response header.

Starting from version 0.8.42, it is possible to specify either a redirect URL (for codes 301, 302, 303, and 307), or the response body text (for other codes). A response body text and redirect URL can contain variables. As a special case, a redirect URL can be specified as a URI local to this server, in which case the full redirect URL is formed according to the request scheme ($scheme) and the server_name_in_redirect and port_in_redirect directives.

In addition, a URL for temporary redirect with the code 302 can be specified as the sole parameter. Such a parameter should start with the “http://”, “https://”, or “$scheme” string. A URL can contain variables.

Only the following codes could be returned before version 0.7.51: 204, 400, 402 — 406, 408, 410, 411, 413, 416, and 500 — 504.

The code 307 was not treated as a redirect until versions 1.1.16 and 1.0.13.

See also the error_page directive.

**rewrite**

**SYNTAX:**

```
rewrite regex replacement [flag];
```

**DEFAULT** —

**CONTEXT:** server, location, if

If the specified regular expression matches a request URI, URI is changed as specified in the replacement string. The rewrite directives are executed sequentially in order of their appearance in the configuration file. It is possible to terminate further processing of the directives using flags. If a replacement string starts with “http://” or “https://”, the processing stops and the redirect is returned to a client.

An optional flag parameter can be one of:

- **last**
  
  stops processing the current set of ngx_http_rewrite_module directives and starts a search for a new location matching the changed URI;

- **break**
  
  stops processing the current set of ngx_http_rewrite_module directives as with the break directive;

- **redirect**
  
  returns a temporary redirect with the 302 code; used if a replacement string does not start with “http://” or “https://”;

- **permanent**
  
  returns a permanent redirect with the 301 code.

The full redirect URL is formed according to the request scheme ($scheme) and the server_name_in_redirect and port_in_redirect directives.

**Example:**
server {
  ...
  rewrite "(/download/(.*)/media/\1\2.*) $1/\3.mp3 last;
  rewrite "(/download/(.*)/audio/\1\2.*) $1/\3.mp3 $2.mp3 last;
  return 403;
  ...
}

But if these directives are put inside the "/download/" location, the last flag should be replaced by break, or otherwise nginx will make 10 cycles and return the 500 error:

```nginx
location /download/ {
  rewrite "(/download/(.*)/media/\1\2.*) $1/\3.mp3 break;
  rewrite "(/download/(.*)/audio/\1\2.*) $1/\3.mp3 $2.ra break;
  return 403;
}
```

If a replacement string includes the new request arguments, the previous request arguments are appended after them. If this is undesired, putting a question mark at the end of a replacement string avoids having them appended, for example:

```nginx
rewrite "/(.*)/show?user=$1? last;
```

If a regular expression includes the "\}" or ";" characters, the whole expressions should be enclosed in single or double quotes.

**rewrite_log**

**SYNTAX:** rewrite_log on | off;

**DEFAULT** off

**CONTEXT:** http, server, location, if

Enables or disables logging of ngx_http_rewrite_module module directives processing results into the error_log at the notice level.

**set**

**SYNTAX:** set $variable value;

**DEFAULT** —

**CONTEXT:** server, location, if

Sets a value for the specified variable. A value can contain text, variables, and their combination.

**uninitialized_variable_warn**

**SYNTAX:** uninitialized_variable_warn on | off;

**DEFAULT** on

**CONTEXT:** http, server, location, if

Controls whether warnings about uninitialized variables are logged.
2.34.3 Internal Implementation

The `ngx_http_rewrite_module` module directives are compiled at the configuration stage into internal instructions that are interpreted during request processing. An interpreter is a simple virtual stack machine.

For example, the directives

```plaintext
location /download/ {
  if ($forbidden) {
    return 403;
  }

  if ($slow) {
    limit_rate 10k;
  }

  rewrite ^/(download/.*)/media/(.*)\..* $ /$1/ mp3 /$2.mp3 break;
}
```

will be translated into these instructions:

```plaintext
variable $forbidden
check against zero
return 403
end of code
variable $slow
check against zero
match of regular expression
  copy "/"
  copy $1
  copy "/mp3/"
  copy $2
  copy ".mp3"
end of regular expression
end of code
```

Note that there are no instructions for the `limit_rate` directive above as it is unrelated to the `ngx_http_rewrite_module` module. A separate configuration is created for the `if` block. If the condition holds true, a request is assigned this configuration where `limit_rate` equals to 10k.

The directive

```plaintext
rewrite ^/(download/.*)/media/(.*)\..* $ /$1/ mp3 /$2.mp3 break;
```

can be made smaller by one instruction if the first slash in the regular expression is put inside the parentheses:

```plaintext
rewrite ^/(download/.*)/media/(.*)\..* $1/ mp3 /$2.mp3 break;
```

The corresponding instructions will then look like this:

```plaintext
match of regular expression
  copy $1
  copy "/mp3/"
  copy $2
  copy ".mp3"
end of regular expression
end of code
```
2.35 Module ngx_http_secure_link_module

2.35.1 Summary

The ngx_http_secure_link_module module (0.7.18) is used to check authenticity of requested links, protect resources from unauthorized access, and limit link lifetime.

The authenticity of a requested link is verified by comparing the checksum value passed in a request with the value computed for the request. If a link has a limited lifetime and the time has expired, the link is considered outdated. The status of these checks is made available in the $secure_link variable.

The module provides two alternative operation modes. The first mode is enabled by the secure_link_secret directive and is used to check authenticity of requested links as well as protect resources from unauthorized access. The second mode (0.8.50) is enabled by the secure_link and secure_link_md5 directives and is also used to limit lifetime of links.

This module is not built by default, it should be enabled with the --with-http_secure_link_module configuration parameter.

2.35.2 Directives

secure_link

SYNTAX: secure_link expression;

DEFAULT —

CONTEXT: http, server, location

Defines a string with variables from which the checksum value and lifetime of a link will be extracted.

Variables used in an expression are usually associated with a request; see example below.

The checksum value extracted from the string is compared with the MD5 hash value of the expression defined by the secure_link_md5 directive. If the checksums are different, the $secure_link variable is set to an empty string. If the checksums are the same, the link lifetime is checked. If the link has a limited lifetime and the time has expired, the $secure_link variable is set to “0”. Otherwise, it is set to “1”. The MD5 hash value passed in a request is encoded in base64url.

If a link has a limited lifetime, the expiration time is set in seconds since Epoch (Thu, 01 Jan 1970 00:00:00 GMT). The value is specified in the expression after the MD5 hash, and is separated by a comma. The expiration time passed in a request is available through the $secure_link_expires variable for a use in the secure_link_md5 directive. If the expiration time is not specified, a link has the unlimited lifetime.
secure_link_md5

SYNTAX:  secure_link_md5 expression;

DEFAULT —

CONTEXT: http, server, location

Defines an expression for which the MD5 hash value will be computed and compared with the value passed in a request.

The expression should contain the secured part of a link (resource) and a secret ingredient. If the link has a limited lifetime, the expression should also contain $secure_link_expires.

To prevent unauthorized access, the expression may contain some information about the client, such as its address and browser version.

Example:

```
location /s/ {
    secure_link $arg_md5,$arg_expires;
    secure_link_md5 "$secure_link_expires$uri$remote_addr secret";

    if ($secure_link = "") {
        return 403;
    }

    if ($secure_link = "0") {
        return 410;
    }

    ...
}
```

The “/s/link?md5=_e4Nc3iduzkWRm01TBBNYw&expires=2147483647” link restricts access to “/s/link” for the client with the IP address 127.0.0.1. The link also has the limited lifetime until January 19, 2038 (GMT).

On UNIX, the md5 request argument value can be obtained as:

```
echo -n '2147483647/s/link127.0.0.1 secret' | \
  openssl md5 -binary | openssl base64 | tr +/ -_ | tr -d =
```

secure_link_secret

SYNTAX:  secure_link_secret word;

DEFAULT —

CONTEXT: location

Defines a secret word used to check authenticity of requested links.

The full URI of a requested link looks as follows:

```
/prefix/hash/link
```

where hash is a hexadecimal representation of the MD5 hash computed for the concatenation of the link and secret word, and prefix is an arbitrary string without slashes.
If the requested link passes the authenticity check, the \$secure_link variable is set to the link extracted from the request URI. Otherwise, the \$secure_link variable is set to an empty string.

Example:

```
location /p/ {
  secure_link_secret secret;
  if (\$secure_link = "") {
    return 403;
  }
  rewrite ^ /secure/\$secure_link;
}
location /secure/ {
  internal;
}
```

A request of “/p/5e814704a28d9bc1914ff19fa0c4a00a/link” will be internally redirected to “/secure/link”.

On UNIX, the hash value for this example can be obtained as:

```
echo -n 'linksecret' | openssl md5 -hex
```

### 2.35.3 Embedded Variables

\$secure_link

The status of a link check. The specific value depends on the selected operation mode.

\$secure_link_expires

The lifetime of a link passed in a request; intended to be used only in the \texttt{secure_link_md5} directive.
2.36 Module ngx_http_session_log_module

2.36.1 Summary

The **ngx_http_session_log_module** module enables logging sessions (that is, aggregates of multiple HTTP requests) instead of individual HTTP requests.

This module is available as part of our [commercial subscription](#).

2.36.2 Example Configuration

The following configuration sets up a session log and maps requests to sessions according to the request client address and *User-Agent* request header field:

```plaintext
session_log_zone /path/to/log format=combined
    zone=one:1m timeout=30s
    md5=$binary_remote_addr$http_user_agent;
location /media/ {
    session_log one;
}
```

2.36.3 Directives

**session_log_format**

**SYNTAX:**  
`session_log_format name string ...`;

**DEFAULT:**  
combined "...

**CONTEXT:** http

Specifies the output format of a log. The value of the `$body_bytes_sent` variable is aggregated across all requests in a session. The values of all other variables available for logging correspond to the first request in a session.

**session_log_zone**

**SYNTAX:**  
`session_log_zone path zone=name:size [format=format]`  
`[timeout=time] [id=id] [md5=md5]`;

**DEFAULT:** —

**CONTEXT:** http

Sets the path to a log file and configures the shared memory zone that is used to store currently active sessions.

A session is considered active for as long as the time elapsed since the last request in the session does not exceed the specified `timeout` (by default, 30 seconds). Once a session is no longer active, it is written to the log.

The `id` parameter identifies the session to which a request is mapped. The `id` parameter is set to the hexadecimal representation of an MD5 hash (for
example, obtained from a cookie using variables). If this parameter is not specified or does not represent the valid MD5 hash, nginx computes the MD5 hash from the value of the `md5` parameter and creates a new session using this hash. Both the `id` and `md5` parameters can contain variables.

The `format` parameter sets the custom session log format configured by the `session_log_format` directive. If `format` is not specified, the predefined “combined” format is used.

**session_log**

```plaintext
SYNTAX:   session_log name | off;
DEFAULT   off
CONTEXT:  http, server, location
```

Enables the use of the specified session log. The special value `off` cancels all `session_log` directives inherited from the previous configuration level.

### 2.36.4 Embedded Variables

The `ngx_http_session_log_module` module supports two embedded variables:

- `$session_log_id`
  - current session ID;
- `$session_log_binary_id`
  - current session ID in binary form (16 bytes).
2.37 Module ngx_http_spdy_module

2.37.1 Summary

The ngx_http_spdy_module module provides experimental support for SPDY. Currently, draft 3.1 of SPDY protocol is implemented.

Before version 1.5.10, draft 2 of SPDY protocol was implemented.

This module is not built by default, it should be enabled with the --with-http_spdy_module configuration parameter.

2.37.2 Known Bugs

The module is experimental, caveat emptor applies.
Current implementation of SPDY protocol does not support “server push”.
In versions prior to 1.5.9, responses in SPDY connections could not be rate limited.

2.37.3 Example Configuration

```plaintext
server {
    listen 443 ssl spdy;
    ssl_certificate server.crt;
    ssl_certificate_key server.key;
    ...
}
```

Note that in order to accept both HTTPS and SPDY connections simultaneously on the same port, OpenSSL library used should support “Next Protocol Negotiation” TLS extension, available since OpenSSL version 1.0.1.

2.37.4 Directives

**spdy_chunk_size**

SYNTAX: `spdy_chunk_size size;`
DEFAULT 8k
CONTEXT: http, server, location
This directive appeared in version 1.5.9.

Sets the maximum size of chunks into which the response body is sliced. A too low value results in higher overhead. A too high value impairs prioritization due to HOL blocking.
spdy_headers.comp

SYNTAX: spdy_headers_comp level;
DEFAULT 0
CONTEXT: http, server

Sets the header compression level of a response in a range from 1 (fastest, less compression) to 9 (slowest, best compression). The special value 0 turns off the header compression.

2.37.5 Embedded Variables

The ngx_http_spdy_module module supports the following embedded variables:

$spdy
   SPDY protocol version for SPDY connections, or an empty string otherwise;
$spdy_request_priority
   request priority for SPDY connections, or an empty string otherwise.
2.38 Module ngx_http_split_clients_module

2.38.1 Summary

The ngx_http_split_clients_module module creates variables suitable for A/B testing, also known as split testing.

2.38.2 Example Configuration

```
http {
    split_clients "${remote_addr}AAA" $variant { 
        0.5% . one;
        2.0% . two;
        * "";
    }
}
server {
    location / {
        index index$ { variant }.html;
    }
}
```

2.38.3 Directives

**split_clients**

**SYNTAX:**  
```
split_clients string $variable { ... }
```

**DEFAULT** —

**CONTEXT:** http

Creates a variable for A/B testing, for example:

```
split_clients "${remote_addr}AAA" $variant { 
    0.5% . one;
    2.0% . two;
    * "";
}
```

The value of the original string is hashed using MurmurHash2. In the example given, hash values from 0 to 21474835 (0.5%) correspond to the value ".one" of the $variant variable, hash values from 21474836 to 107374180 (2%) correspond to the value ".two", and hash values from 107374181 to 4294967295 correspond to the value "" (an empty string).
2.39 Module ngx_http_ssi_module

2.39.1 Summary

The ngx_http_ssi_module module is a filter that processes SSI (Server Side Includes) commands in responses passing through it. Currently, the list of supported SSI commands is incomplete.

2.39.2 Example Configuration

```plaintext
location / {
    ssi on;
    ...
}
```

2.39.3 Directives

ssi

- **SYNTAX:** ssi on | off;
- **DEFAULT:** off
- **CONTEXT:** http, server, location, if in location

Enables or disables processing of SSI commands in responses.

ssi_last_modified

- **SYNTAX:** ssi_last_modified on | off;
- **DEFAULT:** off
- **CONTEXT:** http, server, location

This directive appeared in version 1.5.1.

Allows preserving the Last-Modified header field from the original response during SSI processing to facilitate response caching.

By default, the header field is removed as contents of the response are modified during processing and may contain dynamically generated elements or parts that are changed independently of the original response.

ssi_min_file_chunk

- **SYNTAX:** ssi_min_file_chunk size;
- **DEFAULT:** 1k
- **CONTEXT:** http, server, location

Sets the minimum size for parts of a response stored on disk, starting from which it makes sense to send them using sendfile.
### ssi_silent_errors

**Syntax:**  
ssi_silent_errors on | off;

**Default:** off

**Context:** http, server, location

If enabled, suppresses the output of the “an error occurred while processing the directive” string if an error occurred during SSI processing.

### ssi_types

**Syntax:**  
ssi_types mime-type ... ;

**Default:** text/html

**Context:** http, server, location

Enables processing of SSI commands in responses with the specified MIME types in addition to “text/html”. The special value “*” matches any MIME type (0.8.29).

### ssi_value_length

**Syntax:**  
ssi_value_length length;

**Default:** 256

**Context:** http, server, location

Sets the maximum length of parameter values in SSI commands.

#### 2.39.4 SSI Commands

SSI commands have the following generic format:

```
<!--# command parameter1=value1 parameter2=value2 ... -->
```

The following commands are supported:

**block**

Defines a block that can be used as a stub in the include command. The block can contain other SSI commands. The command has the following parameter:

**name**

block name.

Example:

```
<!--# block name="one" -->
stub
<!--# endblock -->
```

**config**

Sets some parameters used during SSI processing, namely:
errormsg

A string that is output if an error occurs during SSI processing. By default, the following string is output:

```
[an error occurred while processing the directive]
```

timefmt

A format string passed to the **strftime** function used to output date and time. By default, the following format is used:

```
"%A, %d-%b-%Y %H:%M:%S %Z"
```

The “%s” format is suitable to output time in seconds.

echo

Outputs the value of a variable. The command has the following parameters:

- **var**
  - the variable name.

- **encoding**
  - the encoding method. Possible values include none, url, and entity. By default, entity is used.

- **default**
  - a non-standard parameter that sets a string to be output if a variable is undefined. By default, “none” is output. The command

```
<!--# echo var="name" default="no" -->
```

replaces the following sequence of commands:

```
<!--# if expr="$name" --><!--# echo var="name" --><!--# else -->no<!--# endif -->
```

if

Performs a conditional inclusion. The following commands are supported:

```
<!--# if expr="..." -->
...
<!--# elif expr="..." -->
...
<!--# else -->
...
<!--# endif -->
```

Only one level of nesting is currently supported. The command has the following parameter:

- **expr**
  - expression. An expression can be:
• variable existence check:

```
<!--# if expr="$name" -->
```

• comparison of a variable with a text:

```
<!--# if expr="$name = text" -->
<!--# if expr="$name != text" -->
```

• comparison of a variable with a regular expression:

```
<!--# if expr="$name = /text/" -->
<!--# if expr="$name != /text/" -->
```

If a text contains variables, their values are substituted. A regular expression can contain positional and named captures that can later be used through variables, for example:

```
<!--# if expr="$name = /(.+)@(?P<domain>.)+/" -->
<!--# echo var="1" -->
<!--# echo var="domain" -->
<!--# endif -->
```

If a text contains variables, their values are substituted. A regular expression can contain positional and named captures that can later be used through variables, for example:

```
<!--# include file="footer.html" -->
```

includes the result of another request into a response. The command has the following parameters:

file

specifies an included file, for example:

```
<!--# include file="footer.html" -->
```

virtual

specifies an included request, for example:

```
<!--# include virtual="/remote/body.php?argument=value" -->
```

Several requests specified on one page and processed by proxied or FastCGI servers run in parallel. If sequential processing is desired, the wait parameter should be used.

stub

a non-standard parameter that names the block whose content will be output if the included request results in an empty body or if an error occurs during the request processing, for example:

```
<!--# block name="one" -->&nbsp;<!--# endif -->
<!--# include virtual="/remote/body.php?argument=value" stub="one" -->
```

The replacement block content is processed in the included request context.
wait

a non-standard parameter that instructs to wait for a request to fully complete before continuing with SSI processing, for example:

```html
<!--# include virtual="/remote/body.php?argument=value" wait="yes" -->
```

set

a non-standard parameter that instructs to write a successful result of request processing to the specified variable, for example:

```html
<!--# include virtual="/remote/body.php?argument=value" set="one" -->
```

It should be noted that only the results of responses obtained using the ngx_http_proxy_module, ngx_http_memcached_module, ngx_http_fastcgi_module (1.5.6), ngx_http_uwsgi_module (1.5.6), and ngx_http_scgi_module (1.5.6) modules can be written into variables.

set

Sets a value of a variable. The command has the following parameters:

var

the variable name.

value

the variable value. If an assigned value contains variables, their values are substituted.

2.39.5 Embedded Variables

The ngx_http_ssi_module module supports two embedded variables:

$\textit{date\_local}$

current time in the local time zone. The format is set by the config command with the timefmt parameter.

$\textit{date\_gmt}$

current time in GMT. The format is set by the config command with the timefmt parameter.
2.40 Module ngx_http_ssl_module

2.40.1 Summary

The ngx_http_ssl_module module provides the necessary support for HTTPS.

This module is not built by default, it should be enabled with the `--with-http_ssl_module` configuration parameter.

This module requires the OpenSSL library.

2.40.2 Example Configuration

To reduce the processor load it is recommended to

- set the number of worker processes equal to the number of processors,
- enable keep-alive connections,
- enable the shared session cache,
- disable the built-in session cache,
- and possibly increase the session lifetime (by default, 5 minutes):

```
worker_processes auto;

http {
 ...

 server {
     listen 443 ssl;
     keepalive_timeout 70;

     ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2;
     ssl_ciphers AES128-SHA : AES256-SHA : RC4-SHA : DES-CBC3-SHA : RC4-MD5;

     ssl_certificate /usr/local/nginx/conf/cert.pem;
     ssl_certificate_key /usr/local/nginx/conf/cert.key;
     ssl_session_cache shared:SSL:10m;
     ssl_session_timeout 10m;

     ...
 }
```

2.40.3 Directives

ssl

SYNTAX: `ssl on | off;`

DEFAULT `off`

CONTEXT: http, server

Enables the HTTPS protocol for the given virtual server.
It is recommended to use the `ssl` parameter of the `listen` directive instead of this directive.

**ssl_buffer_size**

**SYNTAX:** `ssl_buffer_size size;`

**DEFAULT:** 16k

**CONTEXT:** http, server

This directive appeared in version 1.5.9.

Sets the size of the buffer used for sending data.

By default, the buffer size is 16k, which corresponds to minimal overhead when sending big responses. To minimize Time To First Byte it may be beneficial to use smaller values, for example:

```
ssl_buffer_size 4k;
```

**ssl_certificate**

**SYNTAX:** `ssl_certificate file;`

**DEFAULT:** —

**CONTEXT:** http, server

Specifies a file with the certificate in the PEM format for the given virtual server. If intermediate certificates should be specified in addition to a primary certificate, they should be specified in the same file in the following order: the primary certificate comes first, then the intermediate certificates. A secret key in the PEM format may be placed in the same file.

It should be kept in mind that due to the HTTPS protocol limitations virtual servers should listen on different IP addresses:

```
server {
  listen 192.168.1.1:443;
  server_name one.example.com;
  ssl_certificate /usr/local/nginx/conf/one.example.com.cert;
  ...
}
server {
  listen 192.168.1.2:443;
  server_name two.example.com;
  ssl_certificate /usr/local/nginx/conf/two.example.com.cert;
  ...
}
```

otherwise the first server’s certificate will be issued for the second site.

**ssl_certificate_key**

**SYNTAX:** `ssl_certificate_key file;`

**DEFAULT:** —

**CONTEXT:** http, server
Specifies a file with the secret key in the PEM format for the given virtual server.

**ssl_ciphers**

SYNTAX: `ssl_ciphers ciphers;`

DEFAULT: `HIGH:!aNULL:!MD5`

CONTEXT: `http, server`

Specifies the enabled ciphers. The ciphers are specified in the format understood by the OpenSSL library, for example:

```
```

The full list can be viewed using the “openssl ciphers” command.

The previous versions of nginx used different ciphers by default.

**ssl_client_certificate**

SYNTAX: `ssl_client_certificate file;`

DEFAULT: `—`

CONTEXT: `http, server`

Specifies a file with trusted CA certificates in the PEM format used to verify client certificates and OCSP responses if `ssl_stapling` is enabled.

The list of certificates will be sent to clients. If this is not desired, the `ssl_trusted_certificate` directive can be used.

**ssl_crl**

SYNTAX: `ssl_crl file;`

DEFAULT: `—`

CONTEXT: `http, server`

This directive appeared in version 0.8.7.

Specifies a file with revoked certificates (CRL) in the PEM format used to verify client certificates.

**ssl_dhparam**

SYNTAX: `ssl_dhparam file;`

DEFAULT: `—`

CONTEXT: `http, server`

This directive appeared in version 0.7.2.

Specifies a file with DH parameters for EDH ciphers.
ssl_ecdh_curve
SYNTAX:  ssl_ecdh_curve curve;
DEFAULT: prime256v1
CONTEXT: http, server
This directive appeared in versions 1.1.0 and 1.0.6.

Specifies a curve for ECDHE ciphers.

ssl_prefer_server_ciphers
SYNTAX:  ssl_prefer_server_ciphers on | off;
DEFAULT: off
CONTEXT: http, server

Specifies that server ciphers should be preferred over client ciphers when using the SSLv3 and TLS protocols.

ssl_protocols
SYNTAX:  ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];
DEFAULT: SSLv3 TLSv1 TLSv1.1 TLSv1.2
CONTEXT: http, server

Enables the specified protocols. The TLSv1.1 and TLSv1.2 parameters work only when the OpenSSL library of version 1.0.1 or higher is used.

The TLSv1.1 and TLSv1.2 parameters are supported starting from versions 1.1.13 and 1.0.12, so when the OpenSSL version 1.0.1 or higher is used on older nginx versions, these protocols work, but cannot be disabled.

ssl_session_cache
SYNTAX:  ssl_session_cache off | none | [builtin[size]] [shared:name:size];
DEFAULT: none
CONTEXT: http, server

Sets the types and sizes of caches that store session parameters. A cache can be of any of the following types:

off
the use of a session cache is strictly prohibited: nginx explicitly tells a client that sessions may not be reused.

none
the use of a session cache is gently disallowed: nginx tells a client that sessions may be reused, but does not actually store session parameters in the cache.

builtin
a cache built in OpenSSL; used by one worker process only. The cache size is specified in sessions. If size is not given, it is equal to 20480 sessions. Use of the built-in cache can cause memory fragmentation.

**shared**

a cache shared between all worker processes. The cache size is specified in bytes; one megabyte can store about 4000 sessions. Each shared cache should have an arbitrary name. A cache with the same name can be used in several virtual servers.

Both cache types can be used simultaneously, for example:

```
ssl_session_cache builtin:1000 shared:ssl:10m;
```

but using only shared cache without the built-in cache should be more efficient.

**ssl_session_ticket_key**

SYNTAX: `ssl_session_ticket_key file;`

DEFAULT —

CONTEXT: `http, server`

This directive appeared in version 1.5.7.

Sets a file with the secret key used to encrypt and decrypt TLS session tickets. The directive is necessary if the same key has to be shared between multiple servers. By default, a randomly generated key is used.

If several keys are specified, only the first key is used to encrypt TLS session tickets. This allows to configure key rotation, for example:

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

The file must contain 48 bytes of random data and can be created using the following command:

```
openssl rand 48 > ticket.key
```

**ssl_session_tickets**

SYNTAX: `ssl_session_tickets on | off;`

DEFAULT `on`

CONTEXT: `http, server`

This directive appeared in version 1.5.9.

Enables or disables session resumption through TLS session tickets.
ssl_session_timeout
SYNTAX:  ssl_session_timeout time;
DEFAULT 5m
CONTEXT: http, server

Specifies a time during which a client may reuse the session parameters stored in a cache.

ssl_stapling
SYNTAX:  ssl_stapling on | off;
DEFAULT off
CONTEXT: http, server
This directive appeared in version 1.3.7.

Enables or disables stapling of OCSP responses by the server. Example:

```
ssl_stapling on;
resolver 192.0.2.1;
```

For the OCSP stapling to work, the certificate of the server certificate issuer should be known. If the ssl_certificate file does not contain intermediate certificates, the certificate of the server certificate issuer should be present in the ssl_trusted_certificate file.

For a resolution of the OCSP responder hostname, the resolver directive should also be specified.

ssl_stapling_file
SYNTAX:  ssl_stapling_file file;
DEFAULT —
CONTEXT: http, server
This directive appeared in version 1.3.7.

When set, the stapled OCSP response will be taken from the specified file instead of querying the OCSP responder specified in the server certificate.

The file should be in the DER format as produced by the “openssl ocsp” command.

ssl_stapling_responder
SYNTAX:  ssl_stapling_responder url;
DEFAULT —
CONTEXT: http, server
This directive appeared in version 1.3.7.

Overrides the URL of the OCSP responder specified in the “Authority Information Access” certificate extension.

Only “http://” OCSP responders are supported:
ssl_stapling_responder http://ocsp.example.com/;

ssl_stapling_verify

SYNTAX:   ssl_stapling_verify on | off;
DEFAULT  off
CONTEXT: http, server
This directive appeared in version 1.3.7.

Enables or disables verification of OCSP responses by the server.
For verification to work, the certificate of the server certificate issuer, the root certificate, and all intermediate certificates should be configured as trusted using the ssl_trusted_certificate directive.

ssl_trusted_certificate

SYNTAX:   ssl_trusted_certificate file;
DEFAULT   —
CONTEXT: http, server
This directive appeared in version 1.3.7.

Specifies a file with trusted CA certificates in the PEM format used to verify client certificates and OCSP responses if ssl_stapling is enabled.
In contrast to the certificate set by ssl_client_certificate, the list of these certificates will not be sent to clients.

ssl_verify_client

SYNTAX:   ssl_verify_client on | off | optional | optional_no_ca;
DEFAULT  off
CONTEXT: http, server

Enables verification of client certificates. The verification result is stored in the $ssl_client_verify variable.
The optional parameter (0.8.7+) requests the client certificate and verifies it if the certificate is present.
The optional_no_ca parameter (1.3.8, 1.2.5) requests the client certificate but does not require it to be signed by a trusted CA certificate. This is intended for the use in cases when a service that is external to nginx performs the actual certificate verification. The contents of the certificate is accessible through the $ssl_client_cert variable.

ssl_verify_depth

SYNTAX:   ssl_verify_depth number;
DEFAULT  1
CONTEXT: http, server
Sets the verification depth in the client certificates chain.

### 2.40.4 Error Processing

The `ngx_http_ssl_module` module supports several non-standard error codes that can be used for redirects using the `error_page` directive:

- **495**: an error has occurred during the client certificate verification;
- **496**: a client has not presented the required certificate;
- **497**: a regular request has been sent to the HTTPS port.

The redirection happens after the request is fully parsed and the variables, such as `$request_uri`, `$uri`, `$args` and others, are available.

### 2.40.5 Embedded Variables

The `ngx_http_ssl_module` module supports several embedded variables:

- **`$ssl_cipher`**: returns the string of ciphers used for an established SSL connection;
- **`$ssl_client_cert`**: returns the client certificate in the PEM format for an established SSL connection, with each line except the first prepended with the tab character; this is intended for the use in the `proxy_set_header` directive;
- **`$ssl_client_raw_cert`**: returns the client certificate in the PEM format for an established SSL connection;
- **`$ssl_client_serial`**: returns the serial number of the client certificate for an established SSL connection;
- **`$ssl_client_s_dn`**: returns the “subject DN” string of the client certificate for an established SSL connection;
- **`$ssl_client_i_dn`**: returns the “issuer DN” string of the client certificate for an established SSL connection;
- **`$ssl_client_verify`**: returns the result of client certificate verification: “SUCCESS”, “FAILED”, and “NONE” if a certificate was not present;
- **`$ssl_protocol`**: returns the protocol of an established SSL connection;
- **`$ssl_session_id`**: returns the session identifier of an established SSL connection;
$ssl_session_reused

returns “r” if an SSL session was reused, or “.” otherwise (1.5.11).
2.41 Module ngx_http_status_module

2.41.1 Summary

The ngx_http_status_module module provides access to various status information.

This module is available as part of our commercial subscription.

2.41.2 Example Configuration

```
server {
    location = /status {
        status;
    }
    status_zone example_server;
}
```

The simple monitoring page is shipped with this distribution, accessible as “/status.html” in the default configuration. It requires the location “/status” to be configured as shown above.

2.41.3 Directives

**status**

**SYNTAX:** status;
**DEFAULT:** —
**CONTEXT:** location

The status information will be accessible from the surrounding location.

**status_format**

**SYNTAX:** status_format json;
**SYNTAX:** status_format jsonp [callback];
**DEFAULT:** json
**CONTEXT:** http, server, location

By default, status information is output in the JSON format. Alternatively, data may be output as JSONP. The callback parameter specifies the name of a callback function. The value can contain variables. If parameter is omitted, or the computed value is an empty string, then “ngx_http_status_jsonp_callback” is used.

**status_zone**

**SYNTAX:** status_zone zone;
**DEFAULT:** —
**CONTEXT:** server
Enables collection of virtual server status information in the specified zone. Several virtual servers may share the same zone.

### 2.41.4 Data

The following status information is provided:

**version**
- Version of the provided data set. The current version is 2.

**nginx_version**
- Version of nginx.

**address**
- The address of the server that accepted status request.

**load_timestamp**
- Time of the last reload of configuration, in milliseconds since Epoch.

**timestamp**
- Current time in milliseconds since Epoch.

**connections**
- **accepted**
  - The total number of accepted client connections.
- **dropped**
  - The total number of dropped client connections.
- **active**
  - The current number of active client connections.
- **idle**
  - The current number of idle client connections.

**requests**
- **total**
  - The total number of client requests.
- **current**
  - The current number of client requests.

**server_zones**
- For each status_zone:
  - **processing**
    - The number of client requests that are currently being processed.
  - **requests**
    - The total number of client requests received from clients.
  - **responses**
    - **total**
      - The total number of responses sent to clients.
    - **1xx, 2xx, 3xx, 4xx, 5xx**
      - The number of responses with status codes 1xx, 2xx, 3xx, 4xx, and 5xx.
received
   The total number of bytes received from clients.

sent
   The total number of bytes sent to clients.

upstreams
   For each server in the dynamically configurable group, the following data are provided:

   server
      An address of the server.

   backup
      A boolean value indicating whether the “cache loader” process is still loading data from disk into the cache.

   weight
      Weight of the server.

   state
      Current state, which may be one of “up”, “down”, “unavail”, or “unhealthy”.

   active
      The current number of active connections.

   keepalive
      The current number of idle keepalive connections.

   requests
      The total number of client requests forwarded to this server.

   responses
      total
         The total number of responses obtained from this server.

      1xx, 2xx, 3xx, 4xx, 5xx
         The number of responses with status codes 1xx, 2xx, 3xx, 4xx, and 5xx.

   sent
      The total number of bytes sent to this server.

   received
      The total number of bytes received from this server.

   fails
      The total number of unsuccessful attempts to communicate with the server.

   unavail
      How many times the server became unavailable for client requests (state “unavail”) due to the number of unsuccessful attempts reaching the max_fails threshold.

   health_checks
      checks
         The total number of health check requests made.

   fails
The number of failed health checks.

**unhealthy**
How many times the server became unhealthy (state “unhealthy”).

**last_passed**
Boolean indicating if the last health check request was successful and passed tests.

**downtime**
Total time the server was in the “unavail” and “unhealthy” states.

**downstart**
The time (in milliseconds since Epoch) when the server became “unavail” or “unhealthy”.

**caches**
For each cache (configured by `proxy_cache_path` and the likes):

**size**
The current size of the cache.

**max_size**
The limit on the maximum size of the cache specified in the configuration.

**cold**
Boolean indicating if “cache loader” is still loading data into the cache.

**hits, stale, updating, revalidated**
responses
The total number of responses read from the cache (hits, or stale responses due to `proxy_cache_use_stale` and the likes).

**bytes**
The total number of bytes read from the cache.

**miss, expired, bypass**
responses
The total number of responses not taken from the cache (misses, expires, or bypasses due to `proxy_cache_bypass` and the likes).

**bytes**
The total number of bytes read from the proxied server.

**responses_written**
The total number of responses written to the cache.

**bytes_written**
The total number of bytes written to the cache.


2.42 Module ngx_http_sub_module

2.42.1 Summary

The ngx_http_sub_module module is a filter that modifies a response by replacing one specified string by another.

This module is not built by default, it should be enabled with the --with-http_sub_module configuration parameter.

2.42.2 Example Configuration

```plaintext
location / {
    sub_filter </head> '</head><script language="javascript" src="$script"></script>';
    sub_filter_once on;
}
```

2.42.3 Directives

**sub_filter**

**SYNTAX:** `sub_filter string replacement;`

**DEFAULT:** —

**CONTEXT:** http, server, location

Sets a string to replace and a replacement string. The string to replace is matched ignoring the case. The replacement string can contain variables.

**sub_filter_last_modified**

**SYNTAX:** `sub_filter_last_modified on | off;`

**DEFAULT:** off

**CONTEXT:** http, server, location

This directive appeared in version 1.5.1.

Allows preserving the *Last-Modified* header field from the original response during replacement to facilitate response caching.

By default, the header field is removed as contents of the response are modified during processing.

**sub_filter_once**

**SYNTAX:** `sub_filter_once on | off;`

**DEFAULT:** on

**CONTEXT:** http, server, location

Indicates whether to look for a string to replace once or several times.
sub_filter_types

SYNTAX:  sub_filter_types mime-type ...;

DEFAULT  text/html

CONTEXT:  http, server, location

Enables string replacement in responses with the specified MIME types in addition to “text/html”. The special value “*” matches any MIME type (0.8.29).
2.43 Module ngx_http_upstream_module

2.43.1 Summary

The ngx_http_upstream_module module is used to define groups of servers that can be referenced by the proxy_pass, fastcgi_pass, and memcached_pass directives.

2.43.2 Example Configuration

```
upstream backend {
    server backend1.example.com weight=5;
    server backend2.example.com:8080;
    server unix:/tmp/backend3;
    server backup1.example.com:8080 backup;
    server backup2.example.com:8080 backup;
}

server {
    location / {  
        proxy_pass http://backend;
    }
}
```

Dynamically configurable group, available as part of our commercial subscription:

```
upstream appservers {
    zone appservers 64k;
    server appserv1.example.com weight=5;
    server appserv2.example.com:8080 fail_timeout=5s slow_start=30s;
    server 192.0.2.1 max_fails=3;
    server reserve1.example.com:8080 backup;
    server reserve2.example.com:8080 backup;
}

server {
    location / {  
        proxy_pass http://appservers;
        health_check;
    }
    location /upstream_conf {  
        upstream_conf;
        allow 127.0.0.1;
        deny all;
    }
}
```

2.43.3 Directives

**upstream**

**SYNTAX:** upstream name { ... }

**DEFAULT** —

**CONTEXT:** http
Defines a group of servers. Servers can listen on different ports. In addition, servers listening on TCP and UNIX-domain sockets can be mixed.

Example:

```nginx
upstream backend {
    server backend1.example.com weight=5;
    server 127.0.0.1:8080 max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;
    server backup1.example.com backup;
}
```

By default, requests are distributed between the servers using a weighted round-robin balancing method. In the above example, each 7 requests will be distributed as follows: 5 requests go to `backend1.example.com` and one request to each of the second and third servers. If an error occurs during communication with a server, the request will be passed to the next server, and so on until all of the functioning servers will be tried. If a successful response could not be obtained from any of the servers, the client will receive the result of the communication with the last server.

**server**

**SYNTAX:** `server address [parameters];`

**DEFAULT** —

**CONTEXT:** upstream

Defines the `address` and other `parameters` of a server. The address can be specified as a domain name or IP address, with an optional port, or as a UNIX-domain socket path specified after the “unix:” prefix. If a port is not specified, the port 80 is used. A domain name that resolves to several IP addresses defines multiple servers at once.

The following parameters can be defined:

- `weight=number`
  sets the weight of the server, by default, 1.

- `max_fails=number`
  sets the number of unsuccessful attempts to communicate with the server that should happen in the duration set by the `fail_timeout` parameter to consider the server unavailable for a duration also set by the `fail_timeout` parameter. By default, the number of unsuccessful attempts is set to 1. The zero value disables the accounting of attempts. What is considered an unsuccessful attempt is defined by the `proxy_next_upstream`, `fastcgi_next_upstream`, and `memcached_next_upstream` directives.

- `fail_timeout=time`
  sets
  - the time during which the specified number of unsuccessful attempts to communicate with the server should happen to consider the server unavailable;
and the period of time the server will be considered unavailable. By default, the parameter is set to 10 seconds.

**backup**
marks the server as a backup server. It will be passed requests when the primary servers are unavailable.

**down**
marks the server as permanently unavailable; used along with the ip_hash directive.

Additionally, the following parameters are available as part of our commercial subscription:

**max_conns=number**
limits the maximum number of simultaneous connections to the proxied server (1.5.9). Default value is zero, meaning there is no limit.

**resolve**
monitors changes of the IP addresses that correspond to a domain name of the server, and automatically modifies the upstream configuration without the need of restarting nginx (1.5.12). In order for this parameter to work, the resolver directive must be specified in the http block. Example:

```
http {
    resolver 10.0.0.1;
    upstream u {
        zone ...;
        ... server example.com resolve;
    }
}
```

**route=string**
sets the server route name.

**slow_start=time**
sets the time during which the server will recover its weight from zero to a nominal value, when unhealthy server becomes healthy, or when the server becomes available after a period of time it was considered unavailable. Default value is zero, i.e. slow start is disabled.

If there is only a single server in a group, max_fails, fail_timeout and slow_start parameters are ignored, and such a server will never be considered unavailable.

**zone**

SYNTAX:  zone name size;
DEFAULT  —
CONTEXT:  upstream

Nginx, Inc.  

Defines the *name* and *size* of the shared memory zone that keeps the group’s configuration and run-time state that are shared between worker processes. Such groups allow changing the group membership or modifying the settings of a particular server without the need of restarting nginx. The configuration is accessible via a special location handled by `upstream_conf`.

This directive is available as part of our commercial subscription.

**ip_hash**

**SYNTAX:** `ip_hash;`

**DEFAULT** —

**CONTEXT:** `upstream`

Specifies that a group should use a load balancing method where requests are distributed between servers based on client IP addresses. The first three octets of the client IPv4 address, or the entire IPv6 address, are used as a hashing key. The method ensures that requests from the same client will always be passed to the same server except when this server is unavailable. In the latter case client requests will be passed to another server. Most probably, it will always be the same server as well.

IPv6 addresses are supported starting from versions 1.3.2 and 1.2.2.

If one of the servers needs to be temporarily removed, it should be marked with the `down` parameter in order to preserve the current hashing of client IP addresses.

Example:

```
upstream backend {
    ip_hash;
    server backend1.example.com;
    server backend2.example.com;
    server backend3.example.com down;
    server backend4.example.com;
}
```

Until versions 1.3.1 and 1.2.2, it was not possible to specify a weight for servers using the `ip_hash` load balancing method.

**keepalive**

**SYNTAX:** `keepalive connections;`

**DEFAULT** —

**CONTEXT:** `upstream`

This directive appeared in version 1.1.4.
CHAPTER 2. HTTP SERVER MODULES

Activates the cache for connections to upstream servers.

The `connections` parameter sets the maximum number of idle keepalive connections to upstream servers that are preserved in the cache of each worker process. When this number is exceeded, the least recently used connections are closed.

It should be particularly noted that the `keepalive` directive does not limit the total number of connections to upstream servers that an nginx worker process can open. The `connections` parameter should be set to a number small enough to let upstream servers process new incoming connections as well.

Example configuration of memcached upstream with keepalive connections:

```http
upstream memcached_backend {
    server 127.0.0.1:11211;
    server 10.0.0.2:11211;
    keepalive 32;
}
server {
    ...
    location /memcached/ {
        set $memcached_key $uri;
        memcached_pass memcached_backend;
    }
}
```

For HTTP, the `proxy_http_version` directive should be set to “1.1” and the `Connection` header field should be cleared:

```http
upstream http_backend {
    server 127.0.0.1:8080;
    keepalive 16;
}
server {
    ...
    location /http/ {
        proxy_pass http://http_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        ...
    }
}
```

Alternatively, HTTP/1.0 persistent connections can be used by passing the `Connection: Keep-Alive` header field to an upstream server, though this method is not recommended.

For FastCGI servers, it is required to set `fastcgi_keep_conn` for keepalive connections to work:
When using load balancer methods other than the default round-robin method, it is necessary to activate them before the `keepalive` directive.

SCGI and uwsgi protocols do not have a notion of keepalive connections.

**least_conn**

SYNTAX: `least_conn;`

DEFAULT —

CONTEXT: `upstream`

This directive appeared in versions 1.3.1 and 1.2.2.

Specifies that a group should use a load balancing method where a request is passed to the server with the least number of active connections, taking into account weights of servers. If there are several such servers, they are tried using a weighted round-robin balancing method.

**health_check**

SYNTAX: `health_check [interval=time] [fails=number] [passes=number] [uri=uri] [match=name];`

DEFAULT —

CONTEXT: `location`

Enables periodic health checks of the servers in a `group` referenced in the surrounding location.

The following optional parameters are supported:

- **interval** sets the interval between two consecutive health checks, by default, 5 seconds;
- **fails** sets the number of consecutive failed health checks of a particular server after which this server will be considered unhealthy, by default, 1;
• **passes** sets the number of consecutive passed health checks of a particular server after which the server will be considered healthy, by default, 1;

• **uri** defines the URI used in health check requests, by default, “/”;

• **match** specifies the match block configuring the tests that a response should pass in order for a health check to pass; by default, the response should have status code 2xx or 3xx.

For example,

```
location / {
  proxy_pass http://backend;
  health_check;
}
```

will send “/” requests to each server in the **backend** group every five seconds. If any communication error or timeout occurs, or a proxied server responds with the status code other than 2xx or 3xx, the health check will fail, and the server will be considered unhealthy. Client requests are not passed to unhealthy servers.

Health checks can be configured to test the status code of a response, presence of certain header fields and their values, and the body contents. Tests are configured separately using the **match** directive and referenced in the **match** parameter. For example:

```
http {
  server {
    ...
    location / {
      proxy_pass http://backend;
      health_check match=welcome;
    }
  }

  match welcome {
    status 200;
    header Content-Type = text/html;
    body ~ "Welcome to nginx!";
  }
}
```

This configuration tells that for a health check to pass, the response to a health check request should succeed, have status 200, content type “text/html”, and contain “Welcome to nginx!” in the body.

The server group must reside in the **shared memory**.

If several health checks are defined for the same group of servers, a single failure of any check will make the corresponding server be considered unhealthy.

This directive is available as part of our **commercial subscription**.
match

SYNTAX:  match name {...}

DEFAULT —

CONTEXT: http

Defines the named test set used to verify responses to health check requests.
The following items can be tested in a response:

status 200;
  status is 200
status ! 500;
  status is not 500
status 200 204;
  status is 200 or 204
status ! 301 302;
  status is neither 301 nor 302
status 200-399;
  status is in the range from 200 to 399
status ! 400-599;
  status is not in the range from 400 to 599
status 301-303 307;
  status is either 301, 302, 303, or 307

header Content-Type = text/html;
  header contains Content-Type with value text/html
header Content-Type != text/html;
  header contains Content-Type with value other than text/html
header Connection ~ close;
  header contains Connection with value matching regular expression close
header Connection !~ close;
  header contains Connection with value not matching regular expression close
header Host;
  header contains Host
header ! X-Accel-Redirect;
  header lacks X-Accel-Redirect

body ~ "Welcome to nginx!";
  body matches regular expression “Welcome to nginx!”
body !~ "Welcome to nginx!";
  body does not match regular expression “Welcome to nginx!”

If several tests are specified, the response matches only if it matches all tests.
Only the first 256k of the response body are examined.

Examples:

```nginx
# status is 200, content type is "text/html", # and body contains "Welcome to nginx!"
match welcome {
    status 200;
    header Content-Type = text/html;
    body ~ "Welcome to nginx!";
}

# status is not one of 301, 302, 303, or 307, and header does not have "Refresh;"
match not_redirect {
    status ! 301-303 307;
    header ! Refresh;
}

# status ok and not in maintenance mode
match server_ok {
    status 200-399;
    body !~ "maintenance mode";
}
```

This directive is available as part of our commercial subscription.

**queue**

SYNTAX:  `queue number [timeout=time];`

DEFAULT —

CONTEXT: `upstream`

This directive appeared in version 1.5.12.

If an upstream server cannot be selected immediately while processing a request, and there are the servers in the group that have reached the `max_conns` limit, the request will be placed into the queue. The directive specifies the maximum number of requests that can be in the queue at the same time. If the queue is filled up, or the server to pass the request to cannot be selected within the time period specified in the `timeout` parameter, an error will be returned to the client.

The default value of the `timeout` parameter is 60 seconds.

This directive is available as part of our commercial subscription.
**sticky**

SYNTAX: `sticky cookie name [expires=time] [domain=domain] [path=path];`  
SYNTAX: `sticky route variable . . . ;`  
DEFAULT —  
CONTEXT: `upstream`

This directive appeared in version 1.5.7.

Enables session affinity, which causes requests from the same client to be passed to the same server in a group of servers. Two methods are available, `cookie` and `route`.

When the `cookie` method is used, information about the designated server is passed in an HTTP cookie:

```plaintext
code
upstream backend {
  server backend1.example.com;
  server backend2.example.com;

  sticky cookie srv_id expires=1h domain=.example.com path=/;
}
```

A request that comes from a client not yet bound to a particular server is passed to the server selected by the configured balancing method. Further requests from the same client are passed to the same server. If the designated server cannot process a request, the new server is selected as if the client has not been bound yet.

The first parameter sets the name of the cookie to be set or inspected. Additional parameters may be as follows:

- **expires**
  Sets the time for which a browser should keep the cookie. The special value `max` will cause the cookie to expire on “31 Dec 2037 23:55:55 GMT”. This is the maximum time understood by old browsers. If the parameter is not specified, it will cause the cookie to expire at the end of a browser session.

- **domain**
  Defines the domain for which the cookie is set.

- **path**
  Defines the path for which the cookie is set.

If any parameters are omitted, the corresponding cookie fields are not set.

When the `route` method is used, proxied server assigns client a route on receipt of the first request. All subsequent requests from this client will carry routing information in a cookie or URI. This information is compared with the “route” parameter of the `server` directive to identify the server to which the request should be proxied. If the designated server cannot process a request, the new server is selected by the configured balancing method as if there is no routing information in the request.

The parameters of the `route` method specify variables that may contain routing information. The first non-empty variable is used to find the matching server.
Example:

```plaintext
map $cookie_jsessionid $route_cookie {
  ~.+\.(?P<route>\w+)$ $route;
}

map $request_uri $route_uri {
  "jsessionid"=.+\.(?P<route>\w+)$ $route;
}

upstream backend {
  server backend1.example.com route=a;
  server backend2.example.com route=b;
  sticky route $route_cookie $route_uri;
}
```

Here, the route is taken from the “JSESSIONID” cookie if present in a request. Otherwise, the route from the URI is used.

---

This directive is available as part of our commercial subscription.

**sticky_cookie_insert**

**SYNTAX:** `sticky_cookie_insert name [expires=time] [domain=domain] [path=path];`

**DEFAULT** —

**CONTEXT:** `upstream`

This directive is obsolete since version 1.5.7. An equivalent `sticky` directive with a new syntax should be used instead:

```plaintext
sticky cookie name [expires=time] [domain=domain] [path=path];
```

**upstream_conf**

**SYNTAX:** `upstream_conf;`

**DEFAULT** —

**CONTEXT:** `location`

Turns on the HTTP interface of upstream configuration in the surrounding location. Access to this location should be limited.

Configuration commands can be used to:

- view all primary or backup servers in a group;
- view an individual server;
- modify an individual server;
- add a new server (see the note below);
- remove an individual server.
As noted in the server directive, specifying a server as a domain name may result in several servers being added to the group. Since addresses in a group are not required to be unique, individual servers in a group can be uniquely referenced to only by their ID. IDs are assigned automatically and shown on viewing of the group configuration.

A configuration command consists of parameters passed as request arguments, for example:

```
http://127.0.0.1/upstream_conf?upstream=appservers
```

The following parameters are supported:

- **upstream=** *name*
  - Selects a group. This parameter is mandatory.

- **backup=**
  - If not set, selects primary servers in the group. If set, selects backup servers in the group.

- **id=** *number*
  - Selects an individual primary or backup server in the group.

- **remove=**
  - Removes an individual primary or backup server from the group.

- **add=**
  - Adds a new primary or backup server to the group.

- **server=** *address*
  - Same as the “address” parameter of the server directive.

- **weight=** *number*
  - Same as the “weight” parameter of the server directive.

- **max_fails=** *number*
  - Same as the “max_fails” parameter of the server directive.

- **fail_timeout=** *time*
  - Same as the “fail_timeout” parameter of the server directive.

- **slow_start=** *time*
  - Same as the “slow_start” parameter of the server directive.

- **down=**
  - Same as the “down” parameter of the server directive.

- **up=**
  - The opposite of the “down” parameter of the server directive.

- **route=** *string*
  - Same as the “route” parameter of the server directive.

The first three parameters select a target the command applies to. Without other parameters, the command shows configuration of the selected target.

For example, to view the primary servers in the group, send:

```
http://127.0.0.1/upstream_conf?upstream=appservers
```
To view the backup servers in the group, send:

```
http://127.0.0.1/upstream_conf?upstream=appservers&backup=
```

To view an individual primary server in the group, send:

```
http://127.0.0.1/upstream_conf?upstream=appservers&id=42
```

To view an individual backup server in the group, send:

```
http://127.0.0.1/upstream_conf?upstream=appservers&backup=&id=42
```

To add a new primary or backup server to the group, specify its address in the “server=” parameter. Without other parameters specified, a server will be added with other parameters set to their default values (see the server directive).

For example, to add a new primary server to the group, send:

```
http://127.0.0.1/upstream_conf?add=&upstream=appservers&server=127.0.0.1:8080
```

To add a new backup server to the group, send:

```
http://127.0.0.1/upstream_conf?add=&upstream=appservers&backup=&server=127.0.0.1:8080
```

To add a new primary server to the group, set its parameters to non-default values and mark it as “down”, send:

```
http://127.0.0.1/upstream_conf?add=&upstream=appservers&server=127.0.0.1:8080&weight=2&max_fails=3&fail_timeout=3s&down=
```

To remove an individual primary or backup server from the group, select it with the id= parameter.

For example, to remove an individual primary server from the group, send:

```
http://127.0.0.1/upstream_conf?remove=&upstream=appservers&id=42
```

To remove an individual backup server from the group, send:

```
http://127.0.0.1/upstream_conf?remove=&upstream=appservers&backup=&id=42
```

To modify an individual primary or backup server in the group, select it with the id= parameter.

For example, to modify an individual primary server in the group by marking it as “down”, send:

```
http://127.0.0.1/upstream_conf?upstream=appservers&id=42&down=
```

To modify the address of an individual backup server in the group, send:
To modify other parameters of an individual primary server in the group, send:

```
http://127.0.0.1/upstream_conf?upstream=appservers&id=42&max_fails=3&weight=4
```

This directive is available as part of our commercial subscription.

### 2.43.4 Embedded Variables

The `ngx_http_upstream_module` module supports the following embedded variables:

- **$upstream_addr**
  
  keeps the IP address and port of the server, or the path to the UNIX-domain socket. If several servers were contacted during request processing, their addresses are separated by commas, e.g. “192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock”. If an internal redirect from one server group to another happens, initiated by `X-Accel-Redirect` or `error_page`, then the server addresses from different groups are separated by colons, e.g. “192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock : 192.168.10.1:80, 192.168.10.2:80”.

- **$upstream_cache_status**
  
  keeps the status of accessing a response cache (0.8.3). The status can be either “MISS”, “BYPASS”, “EXPIRED”, “STALE”, “UPDATING”, “REVALIDATED” or “HIT”.

- **$upstream_response_length**
  
  keeps the lengths of responses obtained from the upstream servers (0.7.27); lengths are kept in bytes. Several response lengths are separated by commas and colons like addresses in the `$upstream_addr` variable.

- **$upstream_response_time**
  
  keeps times of responses obtained from upstream servers; times are kept in seconds with a milliseconds resolution. Several response times are separated by commas and colons like addresses in the `$upstream_addr` variable.

- **$upstream_status**
  
  keeps codes of responses obtained from upstream servers. Several response codes are separated by commas and colons like addresses in the `$upstream_addr` variable.

- **$upstream_http...**
  
  keep server response header fields. For example, the `Server` response header field is available through the `$upstream_http_server` variable. The
rules of converting header field names to variable names are the same as for the variables that start with the “$http_” prefix. Only the last server’s response header fields are saved.
2.44 Module ngx_http_userid_module

2.44.1 Summary

The ngx_http_userid_module module sets cookies suitable for client identification. Received and set cookies can be logged using the embedded variables $uid_got and $uid_set. This module is compatible with the mod_uid module for Apache.

2.44.2 Example Configuration

```plaintext
userid on;
userid_name uid;
userid_domain example.com;
userid_path /;
userid_expires 365d;
userid_p3p 'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR STA NID"';
```

2.44.3 Directives

**userid**

**SYNTAX:**  userid on | v1 | log | off;
**DEFAULT**  off
**CONTEXT:**  http, server, location

Enables or disables setting cookies and logging the received cookies:

- **on**
  - enables the setting of version 2 cookies and logging of the received cookies;
- **v1**
  - enables the setting of version 1 cookies and logging of the received cookies;
- **log**
  - disables the setting of cookies, but enables logging of the received cookies;
- **off**
  - disables the setting of cookies and logging of the received cookies.

**userid_domain**

**SYNTAX:**  userid_domain name | none;
**DEFAULT**  none
**CONTEXT:**  http, server, location

Defines a domain for which the cookie is set. The **none** parameter disables setting of a domain for the cookie.
userid_expires

SYNTAX:  userid_expires time | max | off;
DEFAULT off
CONTEXT: http, server, location

Sets a time during which a browser should keep the cookie. The parameter max will cause the cookie to expire on “31 Dec 2037 23:55:55 GMT”. This is the maximum time understood by old browsers. The parameter off will cause the cookie to expire at the end of a browser session.

userid_mark

SYNTAX:  userid_mark letter | digit | = | off;
DEFAULT off
CONTEXT: http, server, location

If the parameter is not off, enables the cookie marking mechanism and sets the character used as a mark. This mechanism is used to add or change userid_p3p and/or a cookie expiration time while preserving the client identifier. A mark can be any letter of the English alphabet (case-sensitive), digit, or the “=” character.

If the mark is set, it is compared with the first padding symbol in the base64 representation of the client identifier passed in a cookie. If they do not match, the cookie is resent with the specified mark, expiration time, and P3P header.

userid_name

SYNTAX:  userid_name name;
DEFAULT uid
CONTEXT: http, server, location

Sets the cookie name.

userid_p3p

SYNTAX:  userid_p3p string | none;
DEFAULT none
CONTEXT: http, server, location

Sets a value for the P3P header field that will be sent along with the cookie. If the directive is set to the special value none, the P3P header will not be sent in a response.

userid_path

SYNTAX:  userid_path path;
DEFAULT /
CONTEXT: http, server, location
Defines a path for which the cookie is set.

**userid_service**

**SYNTAX:** `userid_service number;`

**DEFAULT:** IP address of the server

**CONTEXT:** http, server, location

If identifiers are issued by multiple servers (services), each service should be assigned its own `number` to ensure that client identifiers are unique. For version 1 cookies, the default value is zero. For version 2 cookies, the default value is the number composed from the last four octets of the server’s IP address.

### 2.44.4 Embedded variables

The `ngx_http_userid_module` module supports the following embedded variables:

- `$uid_got`
  - The cookie name and received client identifier.

- `$uid_reset`
  - If the variable is set to a non-empty string that is not “0”, the client identifiers are reset. The special value “log” additionally leads to the output of messages about the reset identifiers to the `error_log`.

- `$uid_set`
  - The cookie name and sent client identifier.
2.45 Module ngx_http_xslt_module

2.45.1 Summary

The ngx_http_xslt_module (0.7.8+) is a filter that transforms XML responses using one or more XSLT stylesheets.

This module is not built by default, it should be enabled with the --with-http_xslt_module configuration parameter.

This module requires the libxml2 and libxslt libraries.

2.45.2 Example Configuration

```conf
location / {
  xml_entities /site/dtd/entities.dtd;
  xslt_stylesheet /site/xslt/one.xslt param=value;
  xslt_stylesheet /site/xslt/two.xslt;
}
```

2.45.3 Directives

**xml_entities**

SYNTAX: `xml_entities path;`

DEFAULT —

CONTEXT: http, server, location

Specifies the DTD file that declares character entities. This file is compiled at the configuration stage. For technical reasons, the module is unable to use the external subset declared in the processed XML, so it is ignored and a specially defined file is used instead. This file should not describe the XML structure. It is enough to declare just the required character entities, for example:

```xml
<!ENTITY nbsp "&#xa0;" >
```

**xslt_last_modified**

SYNTAX: `xslt_last_modified on | off;`

DEFAULT `off`

CONTEXT: http, server, location

This directive appeared in version 1.5.1.

Allows preserving the *Last-Modified* header field from the original response during XSLT transformations to facilitate response caching.

By default, the header field is removed as contents of the response are modified during transformations and may contain dynamically generated elements or parts that are changed independently of the original response.
**xslt_param**

SYNTAX:  
```text
xslt_param parameter value;
```

DEFAULT —

CONTEXT: http, server, location

This directive appeared in version 1.1.18.

Defines the parameters for XSLT stylesheets. The `value` is treated as an XPath expression. The `value` can contain variables. To pass a string value to a stylesheet, the `xslt_string_param` directive can be used.

There could be several `xslt_param` directives. These directives are inherited from the previous level if and only if there are no `xslt_param` and `xslt_string_param` directives defined on the current level.

**xslt_string_param**

SYNTAX:  
```text
xslt_string_param parameter value;
```

DEFAULT —

CONTEXT: http, server, location

This directive appeared in version 1.1.18.

Defines the string parameters for XSLT stylesheets. XPath expressions in the `value` are not interpreted. The `value` can contain variables.

There could be several `xslt_string_param` directives. These directives are inherited from the previous level if and only if there are no `xslt_param` and `xslt_string_param` directives defined on the current level.

**xslt_stylesheet**

SYNTAX:  
```text
xslt_stylesheet stylesheet [parameter=value ... ];
```

DEFAULT —

CONTEXT: location

Defines the XSLT stylesheet and its optional parameters. A stylesheet is compiled at the configuration stage.

Parameters can either be specified separately, or grouped in a single line using the `;` delimiter. If a parameter includes the `;` character, it should be escaped as `"%3A"`. Also, `libxslt` requires to enclose parameters that contain non-alphanumeric characters into single or double quotes, for example:

```
param1='http%3A//www.example.com';param2=value2
```

The parameters description can contain variables, for example, the whole line of parameters can be taken from a single variable:

```
location / {
  xslt_stylesheet /site/xslt/one.xslt
  $arg_xslt_params
  param1='$value1';param2=value2
  param3=value3;
}
```
It is possible to specify several stylesheets. They will be applied sequentially in the specified order.

**xslt_types**

SYNTAX:  

```plaintext
xslt_types mime-type ...;
```

DEFAULT  

```plaintext
text/xml
```

CONTEXT: http, server, location

Enables transformations in responses with the specified MIME types in addition to “text/xml”. The special value “*” matches any MIME type (0.8.29). If the transformation result is an HTML response, its MIME type is changed to “text/html”. 

Chapter 3

Mail server modules

3.1 Module ngx_mail_core_module

3.1.1 Summary

This module is not built by default, it should be enabled with the `--with-mail` configuration parameter.

3.1.2 Example configuration

```plaintext
worker_processes 1;
error_log /var/log/nginx/error.log info;

mail {
  server_name mail.example.com;
  auth_http localhost:9000/cgi-bin/nginxauth.cgi;
  imap_capabilities IMAP4rev1 UIDPLUS IDLE LITERAL+ QUOTA;
  pop3_auth plain apop cram-md5;
  pop3_capabilities LAST TOP USER PIPELINING UIDL;
  smtp_auth login plain cram-md5;
  smtp_capabilities "SIZE 10485760" ENHANCEDSTATUSCODES 8BITMIME DSN;
  xclient off;

  server {
    listen 25;
    protocol smtp;
  }
  server {
    listen 110;
    protocol pop3;
    proxy_pass_error_message on;
  }
  server {
    listen 143;
    protocol imap;
  }
  server {
    listen 587;
    protocol smtp;
  }
}
```
3.1.3 Directives

listen

**SYNTAX:** listen address:port [bind];
**DEFAULT** —
**CONTEXT:** server

Sets the *address* and *port* for the socket on which the server will accept requests. It is possible to specify just the port. The address can also be a hostname, for example:

```plaintext
listen 127.0.0.1:110;
liseten *:110;
liseten 110; # same as *:110
listen localhost:110;
```

IPv6 addresses (0.7.58) are specified in square brackets:

```plaintext
listen [::1]:110;
liseten [::]:110;
```

UNIX-domain sockets (1.3.5) are specified with the "unix:" prefix:

```plaintext
listen unix:/var/run/nginx.sock;
```

The optional `bind` parameter instructs to make a separate `bind` call for a given address:port pair. The fact is that if there are several `listen` directives with the same port but different addresses, and one of the `listen` directives listens on all addresses for the given port (`:port`), nginx will `bind` only to `*:port`. It should be noted that the `getsockname` system call will be made in this case to determine the address that accepted the connection.

Different servers must listen on different address:port pairs.

mail

**SYNTAX:** mail {...}
**DEFAULT** —
**CONTEXT:** main

Provides the configuration file context in which the mail server directives are specified.

protocol

**SYNTAX:** protocol imap | pop3 | smtp;
**DEFAULT** —
**CONTEXT:** server

Sets the protocol for a proxied server. Supported protocols are IMAP, POP3, and SMTP.

If the directive is not set, the protocol can be detected automatically based on the well-known port specified in the `listen` directive:
• imap: 143, 993
• pop3: 110, 995
• smtp: 25, 587, 465

Unnecessary protocols can be disabled using the configuration parameters --without-mail_imap_module, --without-mail_pop3_module, and --without-mail_smtp_module.

**resolver**

*SYNTAX:* resolver address...[valid=time];

*SYNTAX:* resolver off;

*DEFAULT* off

*CONTEXT:* mail, server

Configures name servers used to find the client’s hostname to pass it to the authentication server, and in the XCLIENT command when proxying SMTP. For example:

```
resolver 127.0.0.1 [::1]:5353;
```

An address can be specified as a domain name or IP address, and an optional port (1.3.1, 1.2.2). If port is not specified, the port 53 is used. Name servers are queried in a round-robin fashion.

Before version 1.1.7, only a single name server could be configured. Specifying name servers using IPv6 addresses is supported starting from versions 1.3.1 and 1.2.2.

By default, nginx caches answers using the TTL value of a response. An optional valid parameter allows overriding it:

```
resolver 127.0.0.1 [::1]:5353 valid=30s;
```

Before version 1.1.9, tuning of caching time was not possible, and nginx always cached answers for the duration of 5 minutes.

The special value off disables resolving.

**resolver_timeout**

*SYNTAX:* resolver_timeout time;

*DEFAULT* 30s

*CONTEXT:* mail, server

Sets a timeout for DNS operations, for example:

```
resolver_timeout 5s;
```
server

SYNTAX: server { ... }

DEFAULT —

CONTEXT: mail

Sets the configuration for a server.

server_name

SYNTAX: server_name name;

DEFAULT hostname

CONTEXT: mail, server

Sets the server name that is used:

• in the initial POP3/SMTP server greeting;

• in the salt during the SASL CRAM-MD5 authentication;

• in the EHLO command when connecting to the SMTP backend, if the passing of the XCLIENT command is enabled.

If the directive is not specified, the machine’s hostname is used.

so_keepalive

SYNTAX: so_keepalive on | off;

DEFAULT off

CONTEXT: mail, server

Indicates if the “TCP keepalive” mode should be enabled on the client’s connection (SO_KEEPALIVE socket parameter) when connecting to a proxied server.

timeout

SYNTAX: timeout time;

DEFAULT 60s

CONTEXT: mail, server

Sets the timeout that is used before proxying to the backend starts.
3.2 Module ngx_mail_pop3_module

3.2.1 Directives

**pop3_auth**

**Syntax:** `pop3_auth method ...;
**Default:** plain
**Context:** mail, server

Sets permitted methods of authentication for POP3 clients. Supported methods are:

- **plain**
  - USER/PASS, AUTH/plain, AUTH/login. It is not possible to disable these methods.

- **apop**
  - APOP. In order for this method to work, the password must be stored unencrypted.

- **cram-md5**
  - AUTH CRAM-MD5. In order for this method to work, the password must be stored unencrypted.

**pop3_capabilities**

**Syntax:** `pop3_capabilities extension ...;
**Default:** TOP USER UIDL
**Context:** mail, server

Sets the POP3 protocol extensions list that is passed to the client in response to the `CAPA` command.

The authentication methods specified in the **pop3_auth** and (SASL extension) and **STLS** directives, are automatically added to this list if the **starttls** directive is enabled.

It makes sense to specify the extensions supported by the POP3 backends to which the clients are proxied (if these extensions are related to commands used after the authentication, when nginx transparently proxies the client connection to the backend).

The current list of standardized extensions is published at [www.iana.org](http://www.iana.org).
3.3 Module ngx_mail_imap_module

3.3.1 Directives

imap_auth

SYNTAX:  imap_auth method ...;
DEFAULT  plain
CONTEXT: mail, server

Sets permitted methods of authentication for IMAP clients. Supported methods are:

login
  AUTH=LOGIN
plain
  AUTH=PLAIN
cram-md5
  AUTH=CRAM-MD5. In order for this method to work, the password must be stored unencrypted.

imap_capabilities

SYNTAX:  imap_capabilities extension ...;
DEFAULT  IMAP4 IMAP4rev1 UIDPLUS
CONTEXT: mail, server

Sets the IMAP protocol extensions list that is passed to the client in response to the CAPABILITY command. The authentication methods specified in the imap_auth and STARTTLS directives are automatically added to this list if the starttls directive is enabled.

It makes sense to specify the extensions supported by the IMAP backends to which the clients are proxied (if these extensions are related to commands used after the authentication, when nginx transparently proxies a client connection to the backend).

The current list of standardized extensions is published at www.iana.org.

imap_client_buffer

SYNTAX:  imap_client_buffer size;
DEFAULT  4k|8k
CONTEXT: mail, server

Sets the IMAP commands read buffer size. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform.
3.4 Module ngx_mail_smtp_module

3.4.1 Directives

smtp_auth

SYNTAX: smtp_auth method ...;
DEFAULT login plain
CONTEXT: mail, server

Sets permitted methods of SASL authentication for SMTP clients. Supported methods are:

login
   AUTH LOGIN
plain
   AUTH PLAIN
cram-md5
   AUTH CRAM-MD5. In order for this method to work, the password must be stored unencrypted.

none
   Authentication is not required.

smtp_capabilities

SYNTAX: smtp_capabilities extension ...;
DEFAULT —
CONTEXT: mail, server

Sets the SMTP protocol extensions list that is passed to the client in response to the EHLO command. Authentication methods specified in the smtp_auth directive are automatically added to this list.

It makes sense to specify the extensions supported by the MTA to which the clients are proxied (if these extensions are related to commands used after the authentication, when nginx transparently proxies the client connection to the backend).

The current list of standardized extensions is published at www.iana.org.
3.5 Module ngx_mail_auth_http_module

3.5.1 Directives

auth_http

SYNTAX: auth_http URL;
DEFAULT —
CONTEXT: mail, server

Sets the URL of the HTTP authentication server. The protocol is described below.

auth_http_header

SYNTAX: auth_http_header header value;
DEFAULT —
CONTEXT: mail, server

Appends the specified header to requests to the authentication server. This header can be used as the shared secret to verify that the request comes from nginx. For example:

```
auth_http_header X-Auth-Key "secret_string";
```

auth_http_timeout

SYNTAX: auth_http_timeout time;
DEFAULT 60s
CONTEXT: mail, server

3.5.2 Protocol

The HTTP is used to communicate with the authentication server. The data in the response body is ignored, and the information is passed only in the headers.

Examples of requests and responses:

Request:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: plain # plain/apop/cram-md5
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap # imap/pop3/smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
```

Good response:
CHAPTER 3. MAIL SERVER MODULES 3.5. MODULE NGX_MAIL_AUTH_HTTP_MODULE

HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
Auth-Port: 143

Bad response:

HTTP/1.0 200 OK
Auth-Status: Invalid login or password
Auth-Wait: 3

If there is no Auth-Wait header, an error will be returned and the connection will be closed. The current implementation allocates memory for each authentication attempt. The memory is freed only at the end of a session. Therefore, the number of invalid authentication attempts in a single session must be limited — the server must respond without the Auth-Wait header after 10-20 attempts (the attempt number is passed in the Auth-Login-Attempt header).

When the APOP or CRAM-MD5 are used, a request-response will look as follows.

GET /auth HTTP/1.0
Host: localhost
Auth-Method: apop
Auth-User: user
Auth-Salt: <238188073.1163692009@mail.example.com>
Auth-Pass: auth_response
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org

Good response:

HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
Auth-Port: 143
Auth-Pass: plain-text-pass

If the Auth-User header exists in a response, it overrides the username used to authenticate with the backend.

For the SMTP, the response additionally takes into account the Auth-Error-Code header — if exists, it is used as a response code in case of an error. Otherwise, the 535 5.7.0 code will be added to the Auth-Status.

For example, if the following response is received from the authentication server:

HTTP/1.0 200 OK
Auth-Status: Temporary server problem, try again later
Auth-Error-Code: 451 4.3.0
Auth-Wait: 3

then the SMTP client will receive an error
451 4.3.0 Temporary server problem, try again later

If proxying SMTP does not require authentication, a request will look as follows.

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: none
Auth-User:
Auth-Pass:
Auth-Protocol: smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
Auth-SMTP-Helo: client.example.org
Auth-SMTP-From: MAIL FROM: <>
Auth-SMTP-To: RCPT TO: <postmaster@mail.example.com>
```
3.6 Module ngx_mail_proxy_module

3.6.1 Directives

proxy_buffer

SYNTAX:  proxy_buffer size;
DEFAULT  4k|8k
CONTEXT: mail, server

Sets the size of the buffer used for proxying. By default, the buffer size is equal to one memory page. Depending on a platform, it is either 4K or 8K.

proxy_pass_error_message

SYNTAX:  proxy_pass_error_message on | off;
DEFAULT  off
CONTEXT: mail, server

Indicates whether to pass the error message obtained during the authentication on the backend to the client.

Usually, if the authentication in nginx is a success, the backend cannot return an error. If it nevertheless returns an error, it means some internal error has occurred. In such case the backend message can contain information that should not be shown to the client. However, responding with an error for the correct password is a normal behavior for some POP3 servers. For example, CommuniGatePro informs a user about mailbox overflow or other events by periodically outputting the authentication error. The directive should be enabled in this case.

proxy_timeout

SYNTAX:  proxy_timeout timeout;
DEFAULT  24h
CONTEXT: mail, server

Defines a timeout used after the proxying to the backend had started.

xclient

SYNTAX:  xclient on | off;
DEFAULT  on
CONTEXT: mail, server

Enables or disables the passing of the XCLIENT command with client parameters when connecting to the SMTP backend.

With XCLIENT, the MTA is able to write client information to the log and apply various limitations based on this data.

If XCLIENT is enabled then nginx passes the following commands when connecting to the backend:
- EHLO with the server name
- XCLIENT
- EHLO or HELO, as passed by the client

If the name found by the client IP address points to the same address, it is passed in the NAME parameter of the XCLIENT command. If the name could not be found, points to a different address, or resolver is not specified, the [UNAVAILABLE] is passed in the NAME parameter. If an error has occurred in the process of resolving, the [TEMPUNAVAIL] value is used.

If XCLIENT is disabled then nginx passes the EHLO command with the server name when connecting to the backend if the client has passed EHLO, or HELO with the server name, otherwise.
3.7 Module ngx_mail_ssl_module

3.7.1 Summary

The ngx_mail_ssl_module module provides the necessary support for a mail proxy server to work with the SSL/TLS protocol.

This module is not built by default, it should be enabled with the --with-mail_ssl_module configuration parameter.

This module requires the OpenSSL library.

3.7.2 Directives

ssl

SYNTAX:  `ssl on | off;`
DEFAULT  `off`
CONTEXT: mail, server

Enables the SSL/TLS protocol for the given server.

ssl_certificate

SYNTAX:  `ssl_certificate file;`
DEFAULT  `—`
CONTEXT: mail, server

Specifies a file with the certificate in the PEM format for the given server. If intermediate certificates should be specified in addition to a primary certificate, they should be specified in the same file in the following order: the primary certificate comes first, then the intermediate certificates. A secret key in the PEM format may be placed in the same file.

ssl_certificate_key

SYNTAX:  `ssl_certificate_key file;`
DEFAULT  `—`
CONTEXT: mail, server

Specifies a file with the secret key in the PEM format for the given server.

ssl_prefer_server_ciphers

SYNTAX:  `ssl_prefer_server_ciphers on | off;`
DEFAULT  `off`
CONTEXT: mail, server

Specifies that server ciphers should be preferred over client ciphers when the SSLv3 and TLS protocols are used.
ssl_protocols

SYNTAX:  ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];
DEFAULT SSLv3 TLSv1 TLSv1.1 TLSv1.2
CONTEXT: mail, server

Enables the specified protocols. The TLSv1.1 and TLSv1.2 parameters work only when the OpenSSL library of version 1.0.1 or higher is used.

The TLSv1.1 and TLSv1.2 parameters are supported starting from versions 1.1.13 and 1.0.12 so when the OpenSSL version 1.0.1 or higher is used on older nginx versions, these protocols work, but cannot be disabled.

ssl_session_cache

SYNTAX:  ssl_session_cache off | none | [builtin[:size]] [shared:name:size];
DEFAULT none
CONTEXT: mail, server

Sets the types and sizes of caches that store session parameters. A cache can be of any of the following types:

off
the use of a session cache is strictly prohibited: nginx explicitly tells a client that sessions may not be reused.
none
the use of a session cache is gently disallowed: nginx tells a client that sessions may be reused, but does not actually store session parameters in the cache.
builtin
a cache built in OpenSSL; used by one worker process only. The cache size is specified in sessions. If size is not given, it is equal to 20480 sessions. Use of the built-in cache can cause memory fragmentation.
shared
a cache shared between all worker processes. The cache size is specified in bytes; one megabyte can store about 4000 sessions. Each shared cache should have an arbitrary name. A cache with the same name can be used in several servers.

Both cache types can be used simultaneously, for example:

ssl_session_cache builtin:1000 shared:SSL:10m;

but using only shared cache without the built-in cache should be more efficient.
ssl_session_ticket_key

SYNTAX:  ssl_session_ticket_key file;
DEFAULT —
CONTEXT: mail, server
This directive appeared in version 1.5.7.

Sets a file with the secret key used to encrypt and decrypt TLS session tickets. The directive is necessary if the same key has to be shared between multiple servers. By default, a randomly generated key is used.

If several keys are specified, only the first key is used to encrypt TLS session tickets. This allows to configure key rotation, for example:

| ssl_session_ticket_key current.key;
| ssl_session_ticket_key previous.key; |

The file must contain 48 bytes of random data and can be created using the following command:

openssl rand 48 > ticket.key

ssl_session_timeout

SYNTAX:  ssl_session_timeout time;
DEFAULT 5m
CONTEXT: mail, server

Specifies a time during which a client may reuse the session parameters stored in a cache.

starttls

SYNTAX:  starttls on | off | only;
DEFAULT off
CONTEXT: mail, server

on
  allow usage of the STLS command for the POP3 and the STARTTLS command for the IMAP;
off
  deny usage of the STLS and STARTTLS commands;
only
  require preliminary TLS transition.
Appendix A

Changelog for NGINX Plus

- 1.5.12, released Apr 2, 2014
  - SPDY protocol updated to version 3.1. SPDY/2 is no longer supported.
  - Added PROXY protocol support (the proxy_protocol parameter of the listen directive).
  - IPv6 support added to resolver.
  - DNS names in upstream groups are periodically re-resolved (the resolve parameter of the server directive).
  - Introduced limiting connections to upstream servers (the max_conns parameter) with optional support for connections queue.

- 1.5.7, released Dec 12, 2013
  - Enhanced sticky routing support.
  - Additional status metrics for virtual hosts and cache zones.
  - Cache purge support (also available for FastCGI).
  - Added support for cache revalidation.
  - New module: ngx_http_auth_request_module (authorization based on the result of a subrequest).

- 1.5.3, released Aug 12, 2013
  - Enhanced status monitoring.
  - Load balancing: slow start feature.
  - Added syslog support for both error_log and access_log.
  - Support for Apple HTTP Live Streaming.

- 1.5.0-2, released May 27, 2013
  - Added support for active healthchecks.

- 1.5.0, released May 7, 2013

- 1.3.16, released Apr 19, 2013
  - Added SPDY support.

- 1.3.13, released Feb 22, 2013
  - Added sticky sessions support.
  - Added support for proxying WebSocket connections.
• 1.3.11, released Jan 18, 2013
  – Added base module ngx_http_gunzip_module.
  – New extra module: ngx_http_session_log_module (aggregated session logging).

• 1.3.9-2, released Dec 20, 2012
  – License information updated.
  – End-User License Agreement added to the package.

• 1.3.9, released Nov 27, 2012
  – Added dynamic upstream management feature.
  – PDF documentation bundled into package.

• 1.3.7, released Oct 18, 2012
  – Initial release of NGINX Plus package.
Appendix B

High Availability support

How to set up simple High Availability environment on generic Linux (RHEL/CentOS or Debian/Ubuntu based systems) in an Active/Passive manner:

1. Install nginx-ha package on both nodes by running "yum install nginx-ha" (RHEL/CentOS) or "apt-get install nginx-ha" (Debian/Ubuntu).
2. Run "nginx-ha-setup" on both nodes and follow on-screen instructions. You will need to run this script under root privileges.

The script will guide you through the interactive setup process, enabling an easy way to:

- Install Corosync and Pacemaker packages
- Configure management IP addresses
- Create configuration for Corosync (generate authkey)
- Start Corosync and check connectivity between nodes
- Start Pacemaker and check cluster membership
- Create basic cluster configuration (cluster IP, Active/Passive preferences)

Upon the successful completion, you will have two nodes running NGINX Plus in a highly available Active/Passive pair:

- Active (primary node for nginx and cluster IP address), and
- Passive (standby node for nginx + cluster IP; resources will be transferred to this node on failover from primary).

You can always check your cluster status on both nodes by running:
```
# crm status bynode
```

Further configuration may be required following your specific needs and environment.

Please check Pacemaker documentation for additional details: http://clusterlabs.org/doc/
Appendix C

Legal Notices

At the release moment of this document, there are three versions of NGINX Plus package in distribution:

- NGINX Plus (package name is nginx-plus)
- NGINX Plus/Lua (package name is nginx-plus-lua)
- NGINX Plus/Extras (package name is nginx-plus-extras)

These distributions contain a different set of various open source software components described below.

Open source components included in NGINX Plus, NGINX Plus/Lua and NGINX Plus/Extras are:

- nginx/OSS, distributed under 2-clause BSD license.
  
  Copyright © 2002-2014 Igor Sysoev
  Copyright © 2011-2014 Nginx, Inc.
  All rights reserved.
  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- MurmurHash algorithm, distributed under MIT license.
  
  Copyright © Austin Appleby
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Open source components included in NGINX Plus/Lua and NGINX Plus/Extras are:

- Nginx Development Kit (NDK) module, distributed under BSD license.
  Copyright © Marcus Clyne

- lua-nginx-module, distributed under 2-clause BSD license.
  Copyright © 2009-2014, by Xiaozhe Wang (chaoslawful)
  Copyright © 2009-2014, by Yichun "agentzh" Zhang (章亦春)
  All rights reserved.
  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Open source components included in NGINX Plus/Extra are:

- headers-more-nginx-module, distributed under 2-clause BSD license.
  Copyright © 2009-2014, Yichun "agentzh" Zhang (章亦春)
  Copyright © 2010-2013, Bernd Dorn
  This module is licensed under the terms of the BSD license.
  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
APPENDIX C. LEGAL NOTICES

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- set-misc-nginx-module, distributed under 2-clause BSD license.

Copyright © 2009-2014, Yichun “agentzh” Zhang (章亦春)

This module is licensed under the terms of the BSD license. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Index

accept_mutex, 18
accept_mutex_delay, 18
access_log, 119
add_after_body, 64
add_before_body, 64
add_header, 106
addition_types, 64
aio, 29
alias, 30
allow, 63
ancient_browser, 70
ancient_browser_value, 71
auth_basic, 65
auth_basic_user_file, 65
auth_http, 224
auth_http_header, 224
auth_http_timeout, 224
auth_request, 67
auth_request_set, 67
autoindex, 69
autoindex_exact_size, 69
autoindex localtime, 69
break, 163
charset, 72
charset_map, 73
charset_types, 73
chunked_transfer_encoding, 31
client_body_buffer_size, 31
client_body_in_file_only, 32
client_body_in_single_buffer, 32
client_body_temp_path, 32
client_body_timeout, 32
client_header_buffer_size, 33
client_header_timeout, 33
client_max_body_size, 33
connection_pool_size, 33
create_full_put_path, 76
daemon, 19
dav_access, 75
dav_methods, 76
debug_connection, 19
debug_points, 19
default_type, 34
deny, 63
directio, 34
directio_alignment, 34
disable_symlinks, 34
empty_gif, 77
env, 21
error_log, 20
error_page, 35
etag, 36
events, 21
expires, 106
f4f, 93
f4f_buffer_size, 93
fastcgi_bind, 78
fastcgi_buffer_size, 78
fastcgi_buffering, 79
fastcgi_buffers, 79
fastcgi_busy_buffers_size, 79
fastcgi_cache, 79
fastcgi_cache_bypass, 80
fastcgi_cache_key, 80
fastcgi_cache_lock, 80
fastcgi_cache_lock_timeout, 80
fastcgi_cache_methods, 81
fastcgi_cache_min_uses, 81
fastcgi_cache_path, 81
fastcgi_cache_purge, 82
fastcgi_cache_revalidate, 83
fastcgi_cache_use_stale, 83
fastcgi_cache_valid, 83
fastcgi_catch stderr, 84
fastcgi_connect_timeout, 84
fastcgi_hide_header, 84astcgi_ignore_client_abort, 85
fastcgi_ignore_headers, 85
fastcgi_index, 85
fastcgi_intercept_errors, 86
fastcgi_keep_conn, 86
fastcgi_max_temp_file_size, 86
fastcgi_next_upstream, 86
fastcgi_no_cache, 87
fastcgi_param, 87
fastcgi_pass, 88
fastcgi_pass_header, 88
fastcgi_pass_request_body, 89
fastcgi_pass_request_headers, 89
fastcgi_read_timeout, 89
fastcgi_send_lowat, 89
fastcgi_send_timeout, 89
fastcgi_split_path_info, 90
fastcgi_store, 90
fastcgi_store_access, 91
fastcgi_temp_file_write_size, 91
fastcgi_temp_path, 91
flv, 94
go, 98
goip_city, 95
goip_country, 95
goip_org, 96
goip_proxy, 97
goip_proxy_recursive, 97
gunzip, 101
gunzip_buffers, 101
gzip, 102
gzip_buffers, 102
gzip_comp_level, 102
gzip_disable, 103
gzip_http_version, 103
gzip_min_length, 103
gzip_proxied, 103
gzip_static, 105
gzip_types, 104
gzip_vary, 104
health_check, 201
hls, 108
hls_buffers, 108
hls_fragment, 109
hls_mp4_buffer_size, 109
hls_mp4_max_buffer_size, 109
http, 36
if, 163
if_modified_since, 37
ignore_invalid_headers, 37
image_filter, 110
image_filter_buffer, 111
image_filter_interlace, 111
image_filter_jpeg_quality, 111
image_filter_sharpen, 112
image_filter_transparency, 112
imap_auth, 222
imap_capabilities, 222
imap_client_buffer, 222
include, 21
index, 113
internal, 37
ip_hash, 199
keepalive, 199
keepalive_disable, 38
keepalive_requests, 38
keepalive_timeout, 39
large_client_header_buffers, 39
least_conn, 201
limit_conn, 114
limit_conn_log_level, 115
limit_conn_status, 115
limit_conn_zone, 115
limit_except, 39
limit_rate, 40
limit_rate_after, 40
limit_req, 117
limit_req_log_level, 118
limit_req_status, 118
limit_zone, 116
lingering_close, 40
lingering_time, 41
lingering_timeout, 41
listen, 41, 218
location, 44
lock_file, 22
log_format, 121
log_not_found, 46
log_subrequest, 46

mail, 218
map, 123
map_hash_bucket_size, 124
map_hash_max_size, 125
master_process, 22
match, 203
max_ranges, 46
memcached_bind, 126
memcached_buffer_size, 126
memcached_connect_timeout, 126
memcached_gzip_flag, 127
memcached_next_upstream, 127
memcached_pass, 127
memcached_read_timeout, 128
memcached_send_timeout, 128
merge_slashes, 46
min_delete_depth, 76
modern_browser, 71
modern_browser_value, 71
mp4, 130
mp4_buffer_size, 130
mp4_limit_rate, 130
mp4_limit_rate_after, 131
mp4_max_buffer_size, 130
msie_padding, 47
msie_refresh, 47
multi_accept, 22

open_file_cache, 47
open_file_cache_errors, 48
open_file_cache_min_uses, 48
open_file_cache_valid, 48
open_log_file_cache, 122
optimize_server_names, 48
output_buffers, 48
override_charset, 74

pcre jit, 22
perl, 134
perl_modules, 134
perl_require, 134
perl_set, 135
pid, 23
pop3_auth, 221

pop3_capabilities, 221
port_in_redirect, 49
postpone_output, 49
protocol, 218
proxy_bind, 138
proxy_buffer, 227
proxy_buffer_size, 138
proxy_buffering, 138
proxy_buffers, 139
proxy_busy_buffers_size, 139
proxy_cache, 139
proxy_cache_bypass, 140
proxy_cache_key, 140
proxy_cache_lock, 140
proxy_cache_lock_timeout, 140
proxy_cache_methods, 141
proxy_cache_min_uses, 141
proxy_cache_path, 141
proxy_cache_purge, 142
proxy_cache_revalidate, 143
proxy_cache_use_stale, 143
proxy_cache_valid, 143
proxy_connect_timeout, 144
proxy_cookie_domain, 144
proxy_cookie_path, 145
proxy_headers_hash_bucket_size, 146
proxy_headers_hash_max_size, 146
proxy_hide_header, 146
proxy_http_version, 146
proxy_ignore_client_abort, 146
proxy_ignore_headers, 147
proxy_intercept_errors, 147
proxy_max_temp_file_size, 147
proxy_method, 148
proxy_next_upstream, 148
proxy_no_cache, 149
proxy_pass, 149
proxy_pass_error_message, 227
proxy_pass_header, 151
proxy_pass_request_body, 151
proxy_pass_request_headers, 151
proxy_read_timeout, 151
proxy_redirect, 152
proxy_send_lowat, 153
proxy_send_timeout, 153
proxy_set_body, 154
proxy_set_header, 154
proxy_ssl_ciphers, 154
proxy_ssl_protocols, 155
proxy_ssl_sessionreuse, 155
proxy_store, 155
proxy_store_access, 156
proxy_temp_file_write_size, 156
proxy_temp_path, 157
proxy_timeout, 227
queue, 204
random_index, 158
readahead, 49
real_ip_header, 159
real_ip_recursive, 160
recursive_error_pages, 49
referer_hash_bucket_size, 161
referer_hash_max_size, 161
request_pool_size, 49
reset_timedout_connection, 50
resolver, 50, 219
resolver_timeout, 51, 219
return, 164
rewrite, 165
rewrite_log, 166
root, 51
satisfy, 51
satisfy_any, 52
secure_link, 168
secure_link_md5, 169
secure_link_secret, 169
send_lowat, 52
send_timeout, 52
sendfile, 52
sendfile_max_chunk, 52
server, 53, 197, 220
server_name, 53, 220
server_name_in_redirect, 55
server_names_hash_bucket_size, 55
server_names_hash_max_size, 55
server_tokens, 55
session_log, 172
session_log_format, 171
session_log_zone, 171
set, 166
set_real_ip_from, 159
smtp_auth, 223
smtp_capabilities, 223
so_keepalive, 220
source_charset, 74
spdy_chunk_size, 173
spdy_headers_comp, 174
split_clients, 175
ssi, 176
ssi_last_modified, 176
ssi_min_file_chunk, 176
ssi_silent_errors, 177
ssi_types, 177
ssi_value_length, 177
ssl, 181, 229
ssl_buffer_size, 182
ssl_certificate, 182, 229
ssl_certificate_key, 182, 229
ssl_ciphers, 183
ssl_client_certificate, 183
ssl_crl, 183
ssl_dhparam, 183
ssl_ecdh_curve, 184
ssl_engine, 23
ssl_prefer_server_ciphers, 184, 229
ssl_protocols, 184, 230
ssl_session_cache, 184, 230
ssl_session_ticket_key, 185, 231
ssl_session_tickets, 185
ssl_session_timeout, 186, 231
ssl_stapling, 186
ssl_stapling_file, 186
ssl_stapling_responder, 186
ssl_stapling_verify, 187
ssl_trusted_certificate, 187
ssl_verify_client, 187
ssl_verify_depth, 187
starttls, 231
status, 190
status_format, 190
status_zone, 190
sticky, 205
sticky_cookie_insert, 206
sub_filter, 194
sub_filter_last_modified, 194
sub_filter_once, 194
sub_filter_types, 195

tcp_nodelay, 55
tcp_nopush, 56
timeout, 220
timer_resolution, 23
try_files, 56
types, 58
types_hash_bucket_size, 58
types_hash_max_size, 59
underscores_in_headers, 59
uninitialized_variable_warn, 166
upstream, 196
upstream_conf, 206
use, 24
user, 24
userid, 211
userid_domain, 211
userid_expires, 212
userid_mark, 212
userid_name, 212
userid_p3p, 212
userid_path, 212
userid_service, 213
valid_referers, 161
variables_hash_bucket_size, 59
variables_hash_max_size, 59
worker_aio_requests, 24
worker_connections, 24
worker_cpu_affinity, 24
worker_priority, 25
worker_processes, 25
worker_rlimit_core, 26
worker_rlimit_nofile, 26
worker_rlimit_sigpending, 26
working_directory, 26
xclient, 227
xml_entities, 214
xslt_last_modified, 214
xslt_param, 215
xslt_string_param, 215
xslt_stylesheet, 215
xslt_types, 216
zone, 198